21 世纪高等职业教育计算机系列规划教材

基于 Java 技术的 Web 应用开发

孙 璐 主 编

罗国涛 项巧莲 戴上平 副主编

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书以 ServletAPI2.3 和 JSPl.2 规范为基础，介绍了应用 Java 技术实现 Web 应用开发的相关技术及编程方法，是一本实用教程。

全书共 9 章，内容包括 Web 应用体系架构、HTML、Servlet 技术、JavaScript 技术、JSP 技术、JavaBean 组件技术、数据库应用等。学生在掌握这些知识后，将具有开发电子商务网站、电子政务、企业信息等项目的能力。本书较全面地体现了应用 Java 技术实现 Web 应用开发的发展特性，涉及当前应用广泛的开发规范，结构清晰，应用实例丰富，实现了理论学习和具体应用的充分结合。

本书可作为各大专院校、高等职业院校计算机专业的教材，也可供从事基于 Java 技术的 Web 应用开发的技术人员学习参考。本书同时提供免费下载的教学课件和所有程序源代码。

图书在版编目（CIP）数据

基于 Java 技术的 Web 应用开发/孙璐主编．—北京：电子工业出版社，2009.8
（21 世纪高等职业教育计算机系列规划教材）
ISBN 978-7-121-09286-2

Ⅰ. 基… Ⅱ.孙… Ⅲ. ①JAVA 语言－程序设计－高等学校：技术学校－教材 ②主页制作－程序设计－高等学校：技术学校－教材 Ⅳ.TP312 TP393.092

中国版本图书馆 CIP 数据核字（2009）第 123061 号

策划编辑：徐建军
责任编辑：韩玉宏
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1 092　1/16　印张：14　字数：358 千字
印　　次：2009 年 8 月第 1 次印刷
印　　数：4 000 册　　定价：26.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

基于 Java 技术的 Web 应用开发是 Java 技术教学的第三阶段，该课程的先导课程包括网页制作、Java 语言程序设计、Java 网络编程、数据库应用等。本书介绍了 Web 应用体系架构、HTML、Servlet 技术、JavaScript 技术、JSP 技术、JavaBean 组件技术及数据库应用。学生在掌握这些知识的基础上，将具有利用 Java 开发各种基于 B/S 架构的电子商务与信息化项目的能力。

本书各章相对独立，按进行 B/S 架构项目开发所应有的知识来组织章节。本着“理论适度、重在应用、重在会用”的原则来确定各章内容的深度和广度。理论叙述尽量简单，形象；例子既注重简洁，又注重实用及代码的完整性。并且用一个项目贯穿本书的各个章节，有利于学生提高项目开发的能力，使其能与项目开发无缝接轨。

本书详细讲解了进行 Web 应用开发的所需知识。针对较难理解的问题，从简单到复杂，逐步深入，便于学生掌握。本书分为 9 章。第 1 章和第 2 章介绍 Web 应用体系架构和环境变量的设置。第 3 章介绍贯穿全书的项目的框架及功能模块的划分。第 4 和第 6 章讲述项目前台开发，包括 HTML 和 JavaScript 技术。第 5 章和第 7～9 章讲述项目后台开发，包括 Servlet 技术、JSP 技术、JavaBean 组件技术和数据库应用。

本书面向的读者是具有一定 Java 语言基础的人。本书可作为各大专院校、高等职业院校计算机专业的教材，也可供从事基于 Java 技术的 Web 应用开发的技术人员学习参考。

本书由孙璐、罗国涛组织编写。在编写过程中得到了四川托普信息技术职业学院领导的指导和支持，特别是得到了刘勇军的大力帮助。参加编写工作的还有中南民族大学的项巧莲和华中师范大学的戴上平等。全书由孙璐统稿和审读。在本书的编写过程中得到了各方面的大力支持，在此一并表示感谢。

为了方便教学，请登录 www.huaxin.edu.cn或 www.hxedu.com.cn免费下载与本书配套的教学资料。需要课件的读者也可以与编者联系（sunlugogo@163.com）。

由于编者水平有限，加上时间仓促，书中难免有不当之处，敬请各位同行批评指正，以便在今后的修订中不断改进。

编　者

目　录

第 1 章　Web应用体系架构

📖 **知识点**

- B/S 架构
- HTTP 请求响应模型

📣 **难点**

- B/S 架构

✍ **掌握**

- Web 应用的发展历史
- HTTP 请求响应模型

任务引入

本章主要介绍 Web 应用的体系架构，主要包括 Web 应用的发展历史、HTTP 请求响应模型及动态网页技术介绍与比较。

1.1　Web应用的发展历史

早期的 Web 应用仅仅是对静态信息的访问，服务器根据用户的请求把相应的静态的 HTML 页面传送给客户端进行显示。应用程序不会对请求的信息进行任何的处理。

例 1_1：静态页面的实现。

Ex1_1_1.html

```
<html>
<head>
    <title>Web 应用-----静态显示</title>
</head>
<body>
    <a href="Ex1_1_2.html">本书的写作特点</a>
</body>
</html>
```

Ex1_1_2.html

```
<html>
<head>
    <title>显示静态内容</title>
</head>
<body>
    本书立足于高职高专，语言浅显易懂，对进行 Web 开发的各个环节都予以介绍，并用一个项目贯穿本书的各个章节。理论知识不求全面，以必需、够用为度。重点在于实用技术。
```

```
</body>
</html>
```

后来出现了 Java 的 Applet 小应用程序。当用户发出请求后，Applet 随着页面一起被下载到客户端，Applet 在浏览器的 JVM 中运行，能够提供动态的页面内容。从某种程度说，Applet 扩展了 Web 服务器的功能。

例 1_2：Applet 的应用。

HelloApplet.java

```
import java.awt.*;
import java.applet.*;

public class HelloApplet extends Applet {

    public void init() {
    }

    public void paint(Graphics g) {
        g.drawString("Welcome to Java!!", 50, 60 );
    }
}
```

HelloTest.html

```
<HTML>
<HEAD>
</HEAD>
<BODY >
<CENTER>
<APPLET
    code  = "HelloApplet.class"
    width= 500
    height= 300
    >
</APPLET>
</CENTER>
</BODY>
</HTML>
```

由于 Applet 不能对数据库进行访问，慢慢就出现了在服务器端运行的 Servlet。Servlet 实际上也是一段 Java 程序，它会根据用户提交数据的不同从而响应不同的页面，实现动态 Web 应用。

Servlet 是运行在服务器端的 Web Container 的 Web 容器中，当一个 HTTP 请求传到服务器，Web Server Plugin 会检测这次请求的是静态资源还是动态资源。如果是静态资源，Web Server Plugin 会将请求传递给 Web Server，Web Server 将请求直接映射到某个页面，

不加任何改变地将页面传回给客户端。但如果请求的是动态资源，Web Server Plugin 会将请求传递给 Web Container，Web Container 会调用相应的 Servlet，根据用户提交数据的不同，将动态内容传回给客户端。

例 1_3：Servlet 的应用。

Ex1_3.html

```
<html>
<body bgcolor="cyan" >
<font color="" size="8">
<form action="Ex1_3" method="get">
      <input type="text" name="paraTxt">
       <input type="submit" value="提交">
</form>
</body>
</html>
```

Ex1_3.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Ex5_5 extends HttpServlet
{
      public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException, IOException
      {
            response.setContentType("text/html;charset=GB2312");
            PrintWriter out=response.getWriter();
            out.println("<html>");
            out.println("<head><title>Servlet 的应用</title></head>");
            out.println("<body>");
            out.println("<hr width=80% align=left color=blue>");

            String para=request.getParameter("paraTxt");
            out.print("<br>"+para);
            out.println("<br><br><hr width=80% align=left color=blue>");
            out.println("</body>");
            out.println("</html>");
      }
}
```

在 Ex1_3.html 文件中定义了一个文本框和一个提交按钮，当用户在文本框中输入内容，

单击提交按钮后，服务器端的 Servlet 会得到用户的提交信息，并将其显示出来。根据用户输入的内容不同，显示不同的信息，完成动态页面的显示。

但 Servlet 的缺点在于，如果仅仅使用 Servlet 往往会把业务逻辑和显示逻辑混合到一起。例如，上面的 Servlet 程序将用户信息的获取和显示放到一起，程序的可读性变得很差。这时就出现了 JSP。

JSP 可以用来实现更好的 MVC 模式，也就是分离视图、控制和业务逻辑，这样可以使应用程序的结构更加清晰。视图层一般由界面设计人员和界面程序员来完成，随着用户需求的变化而变化；模型层则由业务逻辑程序员来完成，根据流程的变化而变化；控制层则一般由负责整体控制的程序员来完成，这部分代码比较稳定，一般会实现一个通用的架构。这种模块功能的划分有利于在代码修改过程中进行模块的隔离，而不需要把具有不同功能的代码混杂在一起造成混乱。用 JSP 页面来显示给用户的数据，用 Servlet 控制页面的流程。Servlet 会根据请求信息的不同或应用程序逻辑的设定而调用相应的 JSP 页面。

例 1_4：JSP 页面的实现。

input.html

```
<html>
<head>
<title>input.html</title>
</head>
<body>
<form action="FirstServlet" method="get">
   <input type="text" name="paraTxt">
  <input type="submit" value="提交">
  </form>
</body>
</html>
```

FirstServlet.java

```
package com;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class FirstServlet extends HttpServlet{
      public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException, IOException
      {
          HttpSession session=request.getSession();
          String para=request.getParameter("paraTxt");
         System.out.println(para);
          session.setAttribute("para", para);
```

```
                response.sendRedirect("showInfo.jsp");
            }
        }
```

showInput.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%
String path = request.getContextPath();
String basePath=request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+path+"/";
%>
<html>
  <head>
    <title>My JSP 'showInfo.jsp' starting page</title>
  </head>
  <body>
     用户输入的信息为：<br>
     <%
        String para=(String)session.getAttribute("para");
     %>
     <%=para %>
  </body>
</html>
```

EJB 是企业级的 JavaBean，它提供了对业务逻辑数据的封装。JavaBean 可以通过 Web Container 来访问 EJB。Java 应用程序客户端也可以访问 EJB。

例 1_5：简单的 EJB 应用。

HelloWorld.java

```
package com.foshanshop.ejb3;

public interface HelloWorld {
    public String SayHello(String name);
}
```

HelloWorldBean.java

```
package com.foshanshop.ejb3.impl;
import com.foshanshop.ejb3.HelloWorld;
import javax.ejb.Remote;
import javax.ejb.Stateless;
public class HelloWorldBean implements HelloWorld {

    public String SayHello(String name) {
        return name +"说：你好!世界,这是我的第一个 EJB3 哦。";
```

```
    }
}
```

HelloWorldTest.java

```
package junit.test;
import static org.junit.Assert.*;
import java.util.Properties;
import javax.naming.InitialContext;
import org.junit.BeforeClass;
import org.junit.Test;
import com.foshanshop.ejb3.HelloWorld;
/**
 * 单元测试用例，运行该用例时，你需要把[Jboss 安装目录]/client 下的 jar 加入项目的类路径下。
 *
 */
public class HelloWorldTest {
    protected static HelloWorld helloworld;

    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
        Properties props = new Properties();
        props.setProperty("java.naming.factory.initial",
"org.jnp.interfaces.NamingContextFactory");
        props.setProperty("java.naming.provider.url", "localhost:1099");
        InitialContext ctx = new InitialContext(props);
        helloworld = (HelloWorld)ctx.lookup("HelloWorldBean/remote");
    }

    @Test
    public void testSayHello() {
     String result = helloworld.SayHello("佛山人");
     //System.out.println(result);
        assertEquals("佛山人说：你好!世界,这是我的第一个 EJB3 哦.", result);
    }
}
```

1.2　HTTP请求响应模型

Web 应用是基于 B/S 架构（也就是浏览器/服务器架构）的。应用程序部署在服务器端，客户端通过浏览器访问应用程序。客户端发送 HTTP 请求消息给服务器，服务器将请求传

递给 Web 应用程序，Web 应用程序处理请求，并把响应的 HTML 页面传回给客户端，所以说 HTTP 协议基于请求响应模型。

一个完整的 HTTP 会话过程包括：首先，客户端与 Web 服务器建立连接，客户端向 Web 服务器发送 HTTP 请求消息，Web 服务器处理请求，并将响应消息传送给客户端，这样一个来回后，这个连接就关闭了。

HTTP 超文本传输协议是一个无状态的协议。也就是说，每当客户端访问 Web 服务器上的某个 Web 页面时，都要建立与服务器的一个独立的连接。服务器不保留前一次访问的任何信息。Web 服务器将客户端对某个页面的每次访问都当做相互无关的访问来处理；服务器不会自动保留用户的状态信息。

HTTP 消息包括起始行、题头域和信息体 3 部分。

起始行通常是请求消息的首行，包含 3 个域：HTTP 方法、通用资源标识符和 HTTP 协议版本。尽管有几种 HTTP 方法可以从服务器中检索数据，但是最常用的方法只有 get 和 post 方法。例如，get 请求的请求行如下。

```
get/first.html HTTP/1.0
```

HTTP 请求的题头域可以没有或有多个。请求题头域允许客户端向服务器传递有关请求和客户端本身的一些附加信息。请求消息和响应消息的题头域是相同的。首先是题头域的名称，接着是冒号和值。如果对同一个题头域规定了多个值，则必须用逗号隔开。例如：

```
hosting:localhost:8080
accept-encoding:gzip,deflate
```

一旦服务器接收并处理了请求消息，它就必须向客户端返回一条响应消息。响应消息包含状态行、0 个多个题头域，空行后是一个消息体。状态行包含响应消息所采用的 HTTP 协议版本，之后是响应状态码和状态描述，中间用空格隔开。响应状态码是 3 位数字，用于描述服务器的响应状态。

1*yy*（以 1 开头的 3 位数）：主要是实验性质的。

2*yy*：表明请求成功。例如，200 表明已成功取得了请求页面。

3*yy*：表明在请求成功之前应该采取进一步的行动。

4*yy*：表明浏览器无法满足请求。例如，404 表示请求的页面不存在。

5*yy*：表明服务器出现问题。例如，500 说明服务器内部发生错误。

1.3　动态网页技术介绍与比较

常用的动态网页技术有 ASP（ASP.NET），PHP，CGI，JSP。在早期，动态网页主要采用 CGI（Common Gateway Interface 公用网关接口）技术。CGI 可以用不同的语言编程，如 VB 和 C++等。然后将写好的程序放在服务器上运行，再将其运行结果通过服务器传送到客户端的浏览器上。虽然 CGI 技术发展成熟且功能强大，但编程复杂，效率低下，修改复杂。编写 CGI 页面最常用的语言是 Perl，文件后缀为 pl。

ASP 是 Active Server Page（动态服务器网页）的简称。它将 Web 上的请求转入一个解

释器中，在这个解释器中将所有的 ASP 的 Script 进行分析，再执行。ASP 可以使用 COM 组件，从而达到无限扩展。ASP 编程简单，容易学习。但 ASP 绝大部分在服务器上运行，导致安全性、稳定性、跨平台性受到影响。

PHP 是 Hypertext Preprocessor 的简称。它是一种 HTML 内嵌式的语言，混合了 C、Java、Perl 及 PHP 的新语法。它有比 CGI 更快的执行动态网页的速度，并且对多数数据库提供支持，如 SQL Server，MySQL，Sybase 等。

ASP.NET 的前身是 ASP。ASP 提出了新的不同于传统 CGI 的解决方法，能处理 Web 表单，并产生动态内容。通过特定的 HTML 标记，可以在网页中加入 JavaScript 程序代码，在服务器端执行。ASP.NET 是建立在微软.NET 平台上的网络技术，它可以将页面表现和程序代码分离，并且可以使用多种语言，如 VB，NET，C# 等。

JSP 是使用 Java 语言的 Web 服务器技术，它也提供了在 HTML 代码中混合程序代码的能力。在 JSP 环境下，HTML 负责描述信息的显示样式，而程序代码则用来描述处理逻辑。JSP 是面向 Web 服务器的技术，客户端浏览器不需要任何附加软件的支持。在 JSP 下，代码被编译成 Servlet 并由 Java 虚拟机执行，这种编译操作仅在对 JSP 页面的第一次请求时发生。

习　题

一、单项选择题

以下有关 C/S 和 B/S 架构的说法错误的是（　　）。

A．在 C/S 架构（即客户端/服务器架构）中，有专门的数据库服务器，但客户端还要运行客户端应用程序，这也叫做胖客户端。

B．在 B/S 架构中，客户端在浏览器中只负责表示层逻辑的实现，业务逻辑和数据库都在服务器端运行。也就是说，应用程序部署在服务器端，客户端通过浏览器访问应用程序。

C．通常在 B/S 架构中，客户端发送 HTTP 请求消息给服务器，服务器将请求传递给 Web 应用程序，Web 应用程序处理请求，并把响应的 HTML 页面传回给客户端。

D．Web 应用是基于 C/S 架构的，C/S 架构就是客户端/服务器架构。

二、多项选择题

1．以下对 HTTP 状态码的含义描述正确的是（　　）。

A．200 表示请求成功。

B．400 表示服务器未发现与请求 URL（统一资源定位）匹配的内容。

C．404 表示由于语法错误而导致服务器无法理解请求消息。

D．500 表示服务器内部错误，无法处理请求。

2．MVC 模式的优势有哪些？（　　）

A．MVC 模式使应用程序的结构更加清晰，通过将代码按照层次划分为业务逻辑/数据层、用户界面和应用流程控制这 3 个层次，能够增强代码的稳定性。

B．MVC 模式实现了很好的分布式组件架构。

C．对于项目开发而言，MVC 3 层的分离有利于在项目小组内按照小组成员各自的擅长方面进行分工，有利于 3 个部分并行开发，加快项目进度。

D．MVC 模块功能的划分有利于在代码修改过程中进行模块的隔离，而不需要把具有不同功能的代码混杂在一起造成混乱。

第 2 章　环境变量的设置及开发实例

📖 知识点

- 如何搭建 Web 应用程序的开发环境
- 了解开发环境所必备的硬件和软件

📣 难点

- 如何搭建 Web 应用程序的开发环境

✍掌握

- 以 Tomcat 服务器为中心配置 JSP 和 Servlet 的运行环境

任务引入

要学习 Web 应用程序的开发，首先需要搭建开发环境。所有程序都要在一定的环境下运行。本书的所有例子都是在 MyEclipse6.0+Tomcat6.0+JDK1.6 环境下调试运行的。其中，MyEclipse6.0 为开发平台，表示所有项目都在此平台上运行；Tomcat6.0 为 Java Web 服务器；JDK1.6 为 Java 虚拟机环境设置。

本章主要讲解 Tomcat6.0 的安装步骤、JDK1.6 的安装、环境变量的设置和开发实例。

2.1　环境变量的设置

要学习 Web 应用程序的开发，必须先搭建一个开发环境。本章以 Tomcat 服务器为中心详细介绍如何配置 JSP 和 Servlet 的运行环境。

2.1.1　Tomcat的安装

Tomcat 服务器是一种 Servlet/JSP 容器。它可以用做小型独立服务器，用于测试 Servlet 和 JSP 页面。与所有 Apache 产品一样，Tomcat 是完全免费的。其安装步骤如下。

（1）运行 jakarta-tomcat-6.0.30 按照提示安装。

（2）在如图 2.1 所示窗口中，可以单击“Browse…”按钮选择安装路径，然后单击“Next＞”按钮，得到如图 2.2 所示窗口。

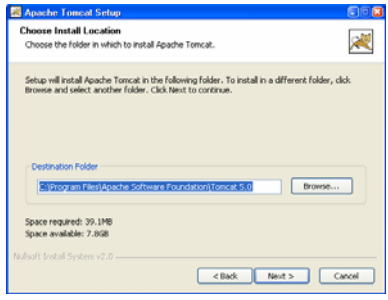


图 2.1　选择安装路径

（3）在如图 2.2 所示窗口中设置密码，可以是空密码；默认端口号为“8080”。单击“Next＞”按钮。

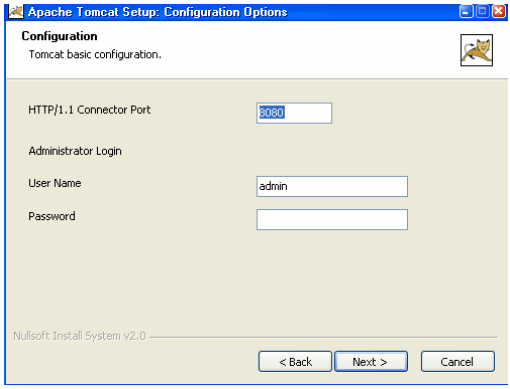


图 2.2　设置密码

（4）选择本计算机上的 JDK 安装路径。如果已经设置了关于 JDK 的系统变量，则安装文件会自动搜索到并显示在本页面上；如果没有，请手动设置。单击“Next＞”按钮。

安装成功后，程序会提示是否启动 Tomcat。若启动，会在系统栏加载图标，图标中心为红色。在图标上单击鼠标右键，选择“Start server”开启服务，这时图标中心应为绿色。

测试服务器：在地址栏中输入“http://localhost:8080”，若显示如图 2.3 所示页面，说明服务器安装成功。

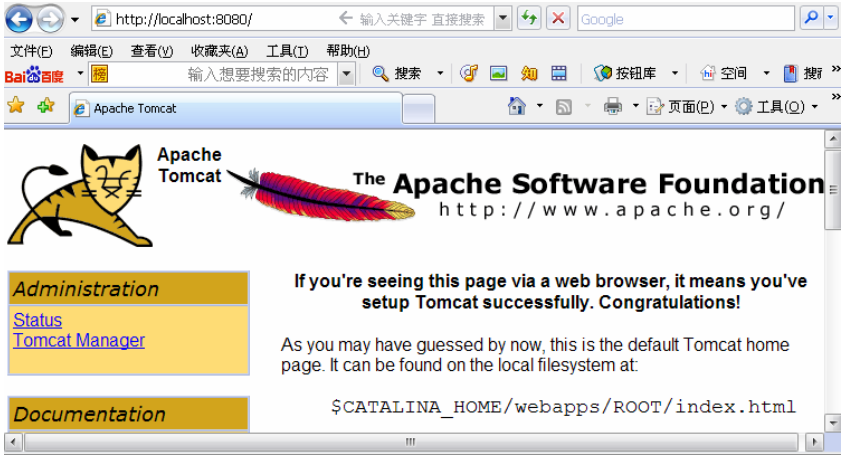


图 2.3　Tomcat 安装成功后的页面

2.1.2　JDK的安装

Java 2 SDK 是 Java 应用程序的基础。JSP 和 Servlet 是基于 Java 技术的，所以要运行 JSP 和 Servlet 必须安装 Java 2 SDK。安装只需要按提示进行即可。安装完成后，需要重新启动计算机。本书的安装路径是 C:\Program Files\Java\jdk1.6.0_02。

2.1.3　设置环境变量

在“我的计算机”图标上单击鼠标右键，弹出菜单，选择“属性”选项，弹出“系统特性”对话框，再选择“高级”选项卡，然后单击“环境变量”按钮，分别添加如下的系统环境变量。

1．设置JDK

（1）定义系统变量 JAVA_HOME，变量值为 JDK 的安装路径“C:\Program Files\Java\jdk1.6.0_02”。

（2）定义系统变量 PATH，变量值为 Java 编译运行时各种命令（如 javac 等）的路径，是在原有值的基础上添加“%JAVA_HOME%\bin”，注意以分号隔开。

（3）定义系统变量 CLASSPATH，变量值为各种类库的路径，为“.;%JAVA_HOME%\lib;”，注意以分号区分各个部分。“.”表示当前目录。

2．设置Tomcat和Servlet

（1）定义系统变量 TOMCAT_HOME，变量值为 Tomcat 的安装路径“C:\Apache Software Foundation\Tomcat 6.0”。

（2）编辑系统变量 CLASSPATH，由于 Servlet 和 JSP 不是 Java2 平台的组成部分，所以必须向编译程序指出 Servlet 类。服务器已经知道 Servlet 类，但编译程序不知道，因此如果不设置 CLASSPATH 而编译 Servlet 或使用 Servlet API 的类将会出现未知类的错误信息。所以必须将 Servlet jar 文件的位置添加到 CLASSPATH 中。变量值为“%TOMCAT_HOME%\common\lib\servlet-api;”。

2.2　开发实例及发布运行

在验证 Tomcat 服务器正常运行后，就可以访问简单的 HTML、Servlet 和 JSP 页面了。

2.2.1　HTML程序

（1）用文本编辑器或 EditPlus，JCreater 编辑 Hello.html 文件。

```
<html>
<title>Hello</title>
<body bgcolor="#ffffff">
        Hello Everyone!
</body>
</html>
```

（2）在 C:\Apache Software Foundation\Tomcat 6.0\webapps 下建立文件夹，本书命名为 test。然后把 WEB_INF 目录复制到 C:\Apache Software Foundation\Tomcat 6.0\webapps\test\ 下。并将 Hello.html 文件复制到其下面。

（3）启动Tomcat，在地址栏中输入“http://localhost:8080/test/Hello.html”（注意需要添加扩展名），应该出现如图 2.4 所示页面。如果出现错误，检查文件名的大小写，检查是否将文件放错了位置，检查Tomcat是否正常启动。

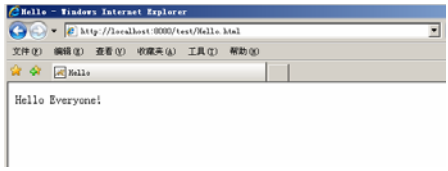


图 2.4　Hello.html 运行页面

2.2.2　Servlet 程序

下面来看 Servlet 的开发、编译、部署和测试。首先需要创建一个自己的开发目录，在其中放置所有的 Servlet 和 JSP。在这里采用上面所建立的目录，即 C:\Apache Software Foundation\Tomcat 6.0\webapps\test\

（1）编写 Servlet 程序，文件名为 Firstpro.java。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Firstpro extends HttpServlet
{
public void doGet(HttpServletRequest request,HttpServletResponse response)throws ServletException,IOException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        out.print("<html><body>");
        out.print("This is the first Servlet program!");
        out.print("</body></html>");
    }
}
```

（2）编译 Firstpro.java，生成 Firstpro.class 文件。

（3）部署与发布。首先将生成的 Firstpro.class 文件复制到 C:\Apache Software Foundation\Tomcat 6.0\webapps\test\WEB_INF\classes 下面；其次需要修改 C:\Apache Software Foundation\Tomcat 6.0\webapps\test\WEB_INF\web.xml 文件，添加如下部分。

```
<servlet>
  <servlet-name>Firstpro</servlet-name>
  <servlet-class>Firstpro</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>Firstpro</servlet-name>
  <url-pattern>/Firstpro</url-pattern>
</servlet-mapping>
```

（4）重新启动Tomcat服务器，在地址栏中输入“http://localhost:8080/test/Firstpro”（注意：文件名后不需要加扩展名）。如果成功执行Servlet，显示如图 2.5 所示页面。如果出现错误，检查名称的大小写，检查class文件的位置，检查系统变量CLASSPATH的设置，以及检查web.xml文件的修改正确与否。

图 2.5　Servlet 程序运行页面

从以上可以看到，Servlet 开发、编译、部署、测试过程包括创建开发目录，设置 CLASSPATH 变量，创建程序代码（Java 文件），编译 Java 文件，编辑 web.xml 文件，以及最后的测试几个环节。

2.2.3　JSP程序

（1）编写 JSP 程序，文件名为 FirstJsp.jsp。JSP 文件的扩展名为 JSP。

```
<html>
<body bgcolor="#b93456">
  <h1> This is the first JSP program! </h1>
        Time is <%= new java.util.Date() %>
</body>
</html>
```

（2）把 FirstJsp.jsp 复制到 C:\Apache Software Foundation\Tomcat 6.0\webapps\test\下面。JSP 文件和 HTML 文件存放的位置是一样的。

（3）重新启动Tomcat服务器，在地址栏中输入“http://localhost:8080/test/FirstJsp.jsp”。注意需要添加扩展名。程序的运行结果如图 2.6 所示。

图 2.6　JSP 程序运行页面

习　题

1．编写程序，在页面上显示：
欢迎进入 Java 的 Web 世界！
现在的时间是：（显示当前的时间）
2．编写程序，在页面上显示：

诗词欣赏

题西林壁
苏轼
横看成岭侧成峰
远近高低各不同
不识庐山真面目
只缘身在此山中

第 3 章　人力资源项目概述

📖 知识点

- 系统概述
- 解决方案
- 系统架构
- 系统功能设计
- 开发和部署

📣 难点

- 人力资源管理系统的业务需求及功能设计

✍ 掌握

- 解决方案的理解
- 系统架构及系统功能设计

项目引入

为了能够更好地理解学习并能够熟练掌握 Java Web 关键技术，如 Servlet 技术、JSP、JavaScript 及数据库连接编程等主要内容，特用实际外包项目人力资源管理系统讲解 Java Web 相关技术的应用。本章主要讲解人力资源管理系统的业务需求分析与功能设计，在后边几章将会结合项目的部分功能需求讲解 Java Web 相关技术在项目中的实际运用。

3.1　系统概述

3.1.1　现状

随着银行业务高速发展，商业银行信息技术部负责服务支撑的系统越来越多，部门人员也在不断增加，目前已达到七十多人的规模，分布在不同的科室。人力资源的丰富可以大大增强技术实力，但同时也对管理工作提出新的挑战。

现有的人力资源管理由综合科负责，目前这项事务管理都是手工操作，无法做到迅速、高效、准确。尤其是现阶段人员较多，每个季度的考评和每位员工的得分，由于计算公式比较复杂，需要耗费大量的时间，极大地影响工作效率。

对每位员工的考评依据，很大一部分来源于其平时表现。具体来说，需要采集一些基础信息，例如，上班工时情况（由请假、休假、加班、值班而来）、工作成果、效率等。但这些信息的采集目前也采用手工完成，费力整理好报表后共享性也差，导致考评人缺乏考评依据，填写考评分数也很难做到公平、公正。

总之，目前的现状是缺乏系统支持，全手工进行所有操作，效率低，统计分析复杂，数据共享性差。

3.1.2 建设目标

将人事考勤纳入进来，流程化管理请假、休假、加班、培训等事务。有了这些基础数据，就可以获得每位员工的工时分布情况。有了这些数据作参照，考评可以做到有据可依，做到公平、公正。

通过系统的建立，实现精细化，管理人员也可以从繁重的手工劳动中解放出来；日常事务电子化后，实现了无纸化办公。

3.2 解决方案

针对商业银行现状，结合需求调研情况，给出如下解决方案。

- 部署：部署在现有的办公自动化平台上。
- 人事档案：具有详细的人事档案记录，除基本信息外，还包含身份证、联系方式、工作经验、证书等。
- 流程化管理日常事务：请假、休假、加班都需要在线填写申请表，批准后生效。
- 工时统计：有了请假、休假、加班这些信息，可以按季度汇总出员工的工时情况，可供全部门查询。
- 自定义考评表单：自定义表单的项目，以及配置考核权重系数。
- 考评结果：自动计算每位员工季度或年度考评得分，但员工只能查看自己的结果，管理员可以查看所有数据。

根据上述方案，其特点如下。

- 自动化：所有报表、数据计算自动完成。
- 随需应变的业务适应能力：可定义考核表单项，配置考核权重系数。
- 兼容性：与现有 OA 系统集成部署，独立运行。

3.3 系统架构

根据该项目功能需求，主要分为 8 个功能模块：系统管理、人员管理、制度管理、补贴统计、事务管理、人员考核、培训管理、消息管理。每个功能模块具体包含哪些功能如图 3.1 所示。

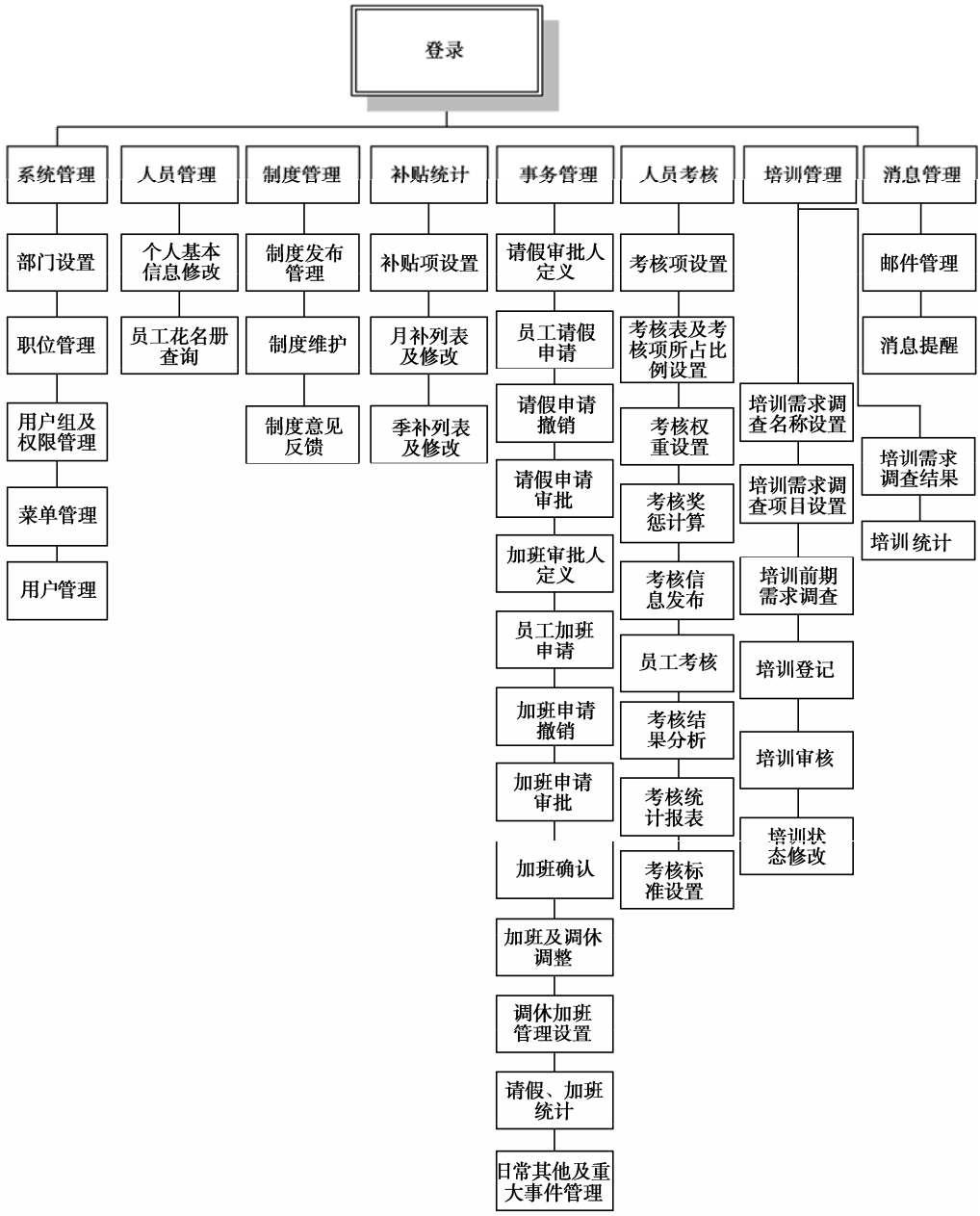


图 3.1　人力资源管理系统架构图

3.4　系统功能设计

前面讲解了系统概述及系统架构，现结合每一个功能模块分析它所包含子模块具体的功能要求。

3.4.1　系统管理

系统管理模块包括如图 3.2 所示的子模块。该模块主要为管理员提供维护整个系统的基础信息。在系统运行之前管理员对这些数据进行设置，在运行时管理员对这些数据按照业务需要进行维护。

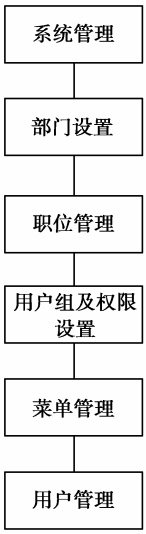


图 3.2　系统管理模块图

在系统管理模块中，每个子模块的具体功能要求如下。

1．部门设置

通过此模块对系统中所存在的机构部门信息进行动态维护。例如，在系统中新增“信息技术部”，并设置“信息技术部”部门为 1 级（最高级）部门；新增“软件 1 科”，并设置“软件 1 科”部门为 2 级部门。

2．职位设置

通过此模块对系统中所存在的员工职位信息进行动态维护，定义职位名称、所属部门、所属部门职位等级。例如，在系统中新增“总经理”职位，并设置“总经理”所属部门为“信息技术部”，职位为“信息技术部”1 级（部门最高级）职位；新增“副总经理”职位，并设置“副总经理”所属部门为“信息技术部”，职位为“信息技术部”2 级职位；新增“科长”职位，并设置“科长”所属部门为“软件 1 科”，职位为“软件 1 科”1 级职位。

3．用户组及权限设置

通过此模块设置系统中各定义用户组动作的权限信息（例如，给某定义用户组增加/删除一个部门设置模块操作权限信息），新建权限，设置权限名称，设置权限动作等。权限是决定用户是否能执行一个动作的唯一依据。

4．菜单管理

通过此模块实现菜单与 URL 的多级动态管理，并通过用户组及权限设置模块与菜单 ID 建立关联，实现用户组的操作权限定义。

5．用户管理

通过此模块对系统的用户信息进行维护，包括①添加新的用户；②为用户设置姓名、登录账号、登录密码、所属角色（管理员/普通用户）、所属部门、所属职位、所属权限组、所属支行；③删除用户（在其他业务表中无数据的可删除）。

其中，根据登录账号判断，若是管理员角色，则打开所有功能可操作页面；若非管理员，则打开可维护登录账号、登录密码、所属部门、所属职位、所属支行的数据页面。

3.4.2　人员管理

人员管理模块包括个人基本信息修改和员工花名册查询两个子模块，如图 3.3 所示。要实现两个子模块的前提是管理员已维护整个系统的人员基本档案信息管理和人员的职位、角色、权限等系统管理设置，在系统运行之前管理员对这些数据进行设置，在运行时管理员对这些数据按照人员调动进行维护。

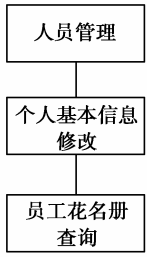


图 3.3　人员管理模块图

在人员管理模块中，每个子模块的具体功能要求如下。

1．个人基本信息修改

在此管理系统中所有涉及员工的最基本信息，包括姓名、性别、学历等。对员工基本信息的操作有新增、修改、删除，同时提供高级查询功能，并对所提出的查询条件统计、生成报表。模块权限给员工本人及管理员。

2．员工花名册查询

员工信息录入系统后，管理员及员工都可以查询员工信息。对员工而言，只能查询自己的信息，而对管理员而言，可以查询所有员工信息。最后，可根据查询条件选择总行、非总行（所有支行）、各支行查询，并导出 Excel 文件。

3.4.3　培训管理

培训管理模块主要包括培训需求名称设置、培训需求调查项目设置、培训前期需求调查、培训需求调查结果、培训登记、培训审核、培训统计、培训状态修改等子模块。培训

管理模块图如图 3.4 所示。

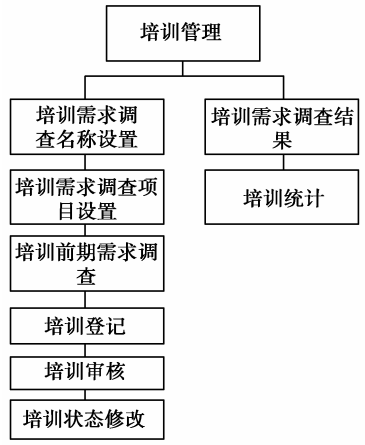


图 3.4　培训管理模块图

在培训管理模块中，每个子模块的具体功能要求如下。

1．培训需求调查名称设置

此功能条件为管理员已经设置好各培训项目的明细，在此基础上根据实际情况设置本次培训需求调查名称。

2．培训需求调查项目设置

前面已经将培训需求调查名称设置好，现把需求调查名称与具体调查项目联系到一起，也就是将培训名称与培训项目建立关系。例如，培训名称为 2008 年 4 季度培训设置，而对应的培训项目可以设置为数据库管理培训、数据库编程培训、中间件培训等。

3．培训前期需求调查

此功能主要用于向培训负责人提供制定培训计划的依据。需要员工在系统中提交个人的培训需求、已培训经历、希望通过培训所达到的目的等信息，培训负责人便可以通过这些信息进行需求的统计，从而为制定具体培训计划提供依据。

4．培训需求调查结果

管理员统计培训需求后，能得到一份统计报告，以作为培训需求调查的结果，然后再将结果导出 Excel 表。

5．培训登记

通过此模块将培训名称、培训地点、培训时间、培训费用、培训层次、参加人员、是否需要领导审核进行明确的登记。

6．培训审核

根据前面的培训登记信息，对于需要审核通过才能进行的培训，审核人员填写审核意见。

7．培训统计

通过此模块可以根据培训项目的名称、培训形式、培训时间、培训层次进行查询，并可以统计培训人数生成统计表。

8．培训状态修改

管理员可根据员工的实际培训情况，修改各员工的培训状态。例如，员工的培训状态可以为已经培训、申请待审核、因故未培训等状态。

3.4.4　补贴统计

补贴统计模块主要包括补贴项设置、月补列表及修改、季补列表及修改等子模块。每个子模块的具体功能要求如下。

1．补贴项设置

通过此模块添加或删除所需要发放的补贴的名称及金额，以供设置每位员工的补贴。

2．月补列表及修改

通过此模块按月为每位员工手工添加修改月补贴，以补贴项乘以 N 次的形式显示。可以按员工姓名、部门、职务、时间等信息给员工设置月补信息。

3．季补列表及修改

按季为每位员工设置季补贴。由于该补贴可能一年只设置一次，所以提供一个沿用功能，例如，一季度的补贴设置沿用到二、三季度。

3.4.5　人员考核

人员考核模块包括考核项设置、考核表及考核项所占比例设置、考核权重设置、考核奖惩计算、考核信息发布、员工考核、考核结果分析、考核统计报表、考核标准设置子模块。每个子模块的具体功能要求如下。

1．考核项设置

通过此模块对考核项进行管理。包括添加、删除、修改、查询考核项。

2．考核表及考核项所占比例设置

考核表也就是被考核人与考核规则关系的总称。通过该模块设置被考核人对应的考核规则，通过考核规则确定考核项和考核项对应的权重，也就是被考核人对应哪个考核项及该考核项的比例设置。

3．考核权重设置

前面已经设置了考核项，现在需要对每一个考核项进行权重的设置，也就是每一个考核项占整个考核的比例。

4．考核奖惩计算

根据前面考核项及考核项权重的设置，通过打分，系统能够得到每个被考核人最后的总分。然后需要根据当前奖惩规则针对被考核人的日常工作情况及重大事件情况等计算该被考核人的奖惩分数，最后得到一个该被考核人的最终考核分数。所以系统将提供给管理员设置奖惩规则的功能。

5．考核信息发布

前面设置了被考核人与考核规则的关系，现对员工与对应规则进行打分。在打分时可以查看评分规则。然后将打分的情况给予发布，让所有员工能够查到自己的得分情况。

6．员工考核

前面进行了考核表及考核项所占比例的设置。现可以修改被考核人与规则之间的关系。也可以通过员工姓名、部门、职务等信息查询出员工信息，然后修改或删除员工信息。

7．考核结果分析

员工通过自己的帐号登录系统，查询自己的考核结果。例如，可以看见自己的考核总分数、考核的奖惩分数、考核的等级等信息。

8．考核统计报表

管理员可以根据考核的名称、科室、职位、考核等级、考核时间等信息查看所有员工的考核结果，能按考核时间生成考核报表，并能进行 Excel 文件导出操作。

9．考核标准设置

对于用户需要针对每一年能够动态制定考核标准（即能够通过分数查询到对应的考核等级）这一需求，管理员能够在后台动态地设置年度考核等级。此考核等级分为优秀、合格、基本合格、不合格等等级。

3.4.6　事务管理

事务管理模块的主要功能为管理系统申请、审批流程，如请假、加班等。管理员可以分别对这些流程进行定义，并让系统按照此流程执行。相应地，员工通过此模块提交所需的申请流程，并通过此模块了解流程的状态变化，以便及时快速地观察到流程的执行情况。管理员还可以通过此模块生成请假、加班等的统计报表。事务管理模块图如图 3.5 所示。

图 3.5　事务管理模块图

在事务管理模块中，每个子模块的具体功能要求如下。

1．请假审批人定义

请假审批人定义流程图如 3.6 所示。

管理员通过此模块对系统中的请假申请流程审批环节进行细节上的定义，如定义流程所需要经过的四级审批人。

系统管理员可以通过增加 “半天以内”、“半天以上”等条件进行自定义设置，以达到动态管理流程判断条件的目的。

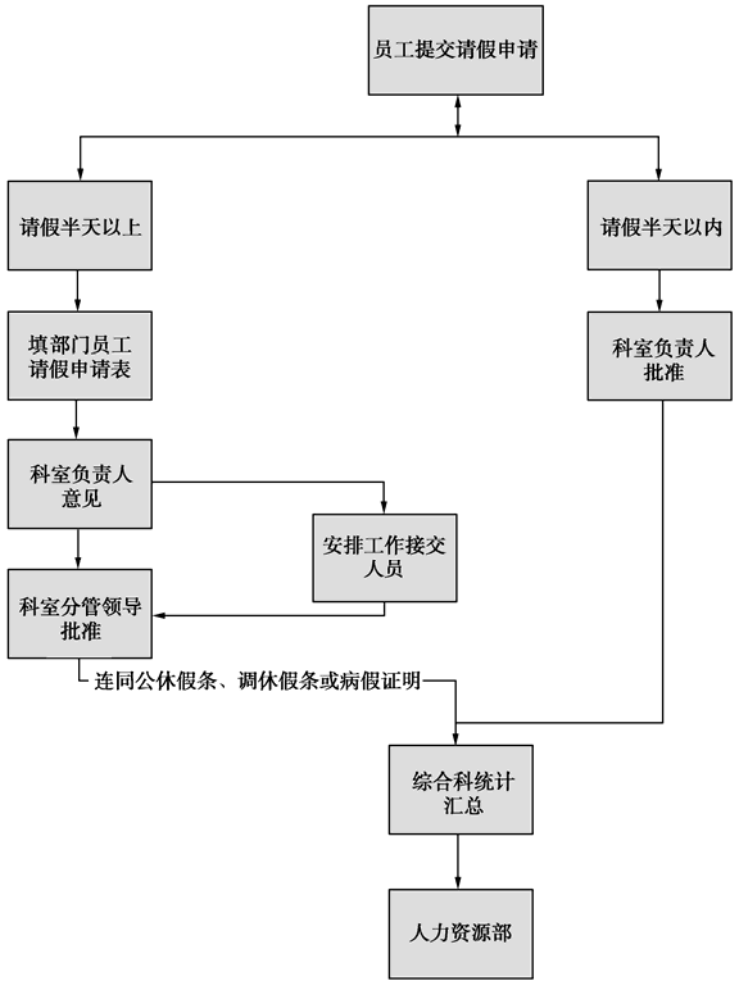


图 3.6　请假审批人定义流程图

2．员工请假申请

每位员工在登录到系统后，按照管理员所设定的请假申请流程，将填写请假申请。申请内容包括请假天数、请假时间、选择审批人、请假类别、请假原因等信息。员工将请假申请提交到系统，等待上级的审批，完成后员工能在系统中看到当前审批的结果。该模块用例图如图 3.7 所示。

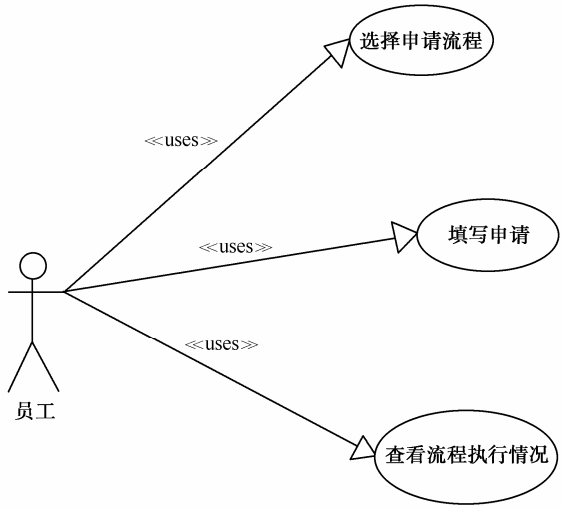


图 3.7　员工请假申请用例图

3．请假申请撤销

员工填写完请假申请后，若还未被审批，则可以撤销其请假申请。

4．请假申请审批

若员工填写完请假申请后，未撤销，又未被审批，则根据请假审批人定义，属于该流程定义中的审批人登录系统后即可对属于他审批的员工进行审批。在请假申请状态中要为未审批，在审批人审批时可以查看员工的姓名、请假天数、请假时间、请假类别、请假原因等信息，然后确定是否同意。该模块用例图如图 3.8 所示。

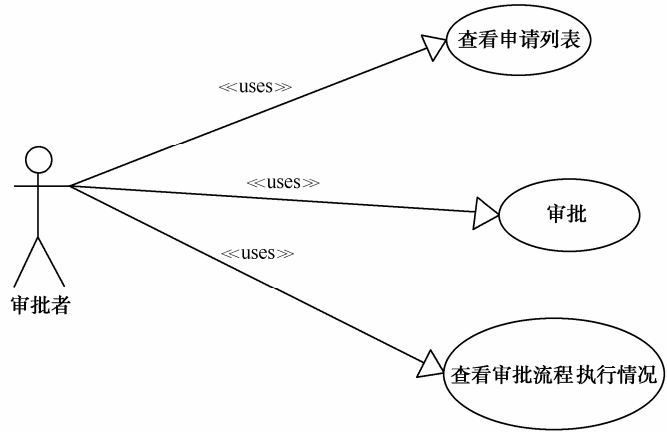


图 3.8　请假申请审批用例图

5．加班审批人定义

管理员通过此模块对系统中的加班申请流程审批环节进行细节上的定义，如为每位员工定义加班申请流程所需要的审核人。在该定义中，主要定义审批人与流程之间的关系。加班审批人定义流程图如图 3.9 所示。

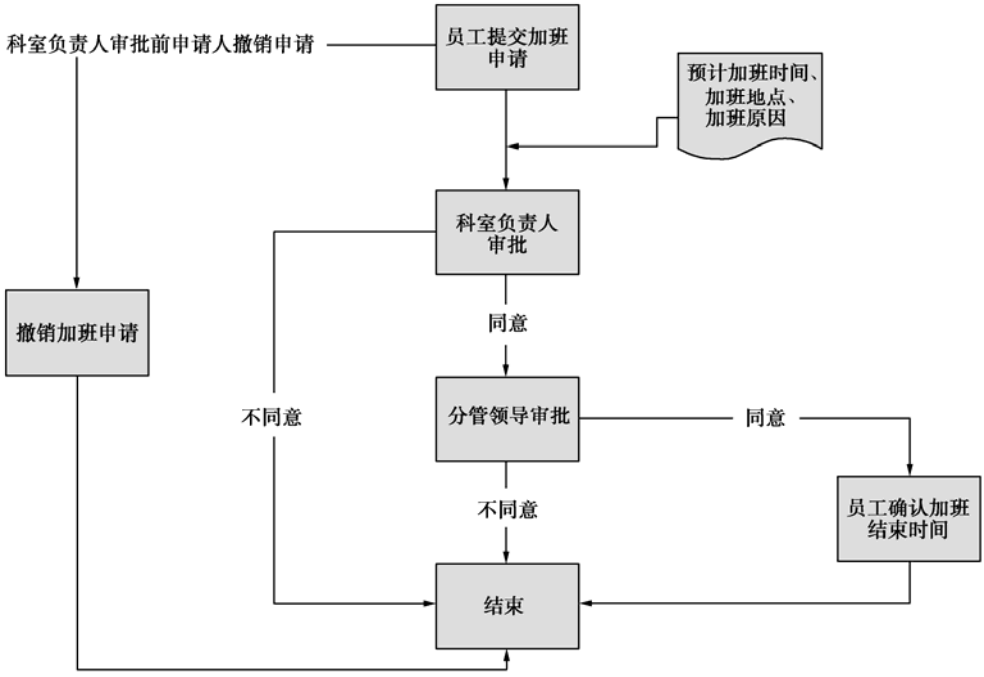


图 3.9　加班审批人定义流程图

6．员工加班申请

每位员工在登录到系统后，按照管理员所设定的加班申请流程，将填写加班申请，并提交到系统，等待上级的审批，完成后员工能在系统中看到当前审批的结果。在填写申请时，需要填写加班时间、加班地点、加班原因等信息。

7．加班申请撤销

员工填写完加班申请后，若未被审批，则可以撤销其加班申请。

8．加班申请审批

若员工填写完加班申请后，未撤销，又未被审批，则根据加班审批人定义，属于该流程定义中的审批人登录系统后即可对属于他审批的员工进行审批。在加班申请状态中要为未审批，在审批人审批时可以查看员工的姓名、加班小时数、加班时间、加班原因等信息，然后确定是否同意。

9．加班确认

根据加班审批人定义流程，当员工的加班申请已经被所有审批人审批同意后，员工需要对其加班申请进行再次确认。

10．加班及调休调整

用于灵活调整员工的每个加班申请单所产生的加班、误餐、调休数据，并记录调整过程。

11．调休加班管理设置

在此模块中，需要设置加班、调休有效开始时间，以便于进行调休天数计算。

12．请假、加班统计

管理员能够对员工通过系统提交的请假、加班等信息进行自定义的统计，并生成相应的报表。例如，在加班统计当中，可以根据加班人姓名、岗位、部门、时间来查询满足条件的员工的加班总天数，并可以查看每位员工加班的具体统计细节。

13．日常其他及重大事件管理

该模块包括日常其他及重大事件增加和日常其他及重大事件维护两个功能。

日常其他及重大事件包括了一级事务类型：旷工、迟到、日常违规（二级事务类型：被投诉、未提交周报、未打扫清洁等，由码表进行维护新增类型）、重大事件（三级事务类型：全行通报批评、全科通报批评等，由码表进行维护新增类型）。日常其他及重大事件记录的奖惩标准参考人员考核模块中的考核奖惩计算子模块。在日常其他及重大事件维护子模块中，可以根据姓名、部门、职务、时间、事务类型等条件查询员工重大事件，并且可以对该重大事件进行删除、修改等。

3.4.7　制度管理

制度管理模块主要包括制度发布管理、制度维护、制度意见反馈子模块。每个子模块的具体功能要求如下。

1．制度发布管理

通过此模块可由管理员将已经批准执行的制度录入系统，单击“发布”按钮对需要执行制度的科室发布制度信息，同时，该模块还提供修改和查看意见的功能。例如，对制度名称、制度编号、执行时间、适用范围、制度状态、制度内容进行发布添加。

2．制度维护

根据制度名称、适用范围、制度状态、制度编号、执行时间等查询条件，查出制度列表。可修改制度及查看意见。

3．制度意见反馈

该功能供员工查看各种制度，并发表相应的意见。

3.4.8　消息管理

消息管理模块包括邮件管理和消息提醒子模块。每个子模块的功能要求如下。

1．邮件管理

通过该模块管理邮件信息，可以向其他人发送邮件，然后可以对已发送邮件进行编辑、删除等。

2．消息提醒

在请假、加班、考核需要审批人审批时，该模块能够给予审批人以提示，便于审批人及时审批信息。

3.5　开发和部署

一个项目的实现，离不开环境的搭建。环境包括软件环境和硬件环境。该系统的软件

环境和硬件环境要求如下。

1．软件环境

- 开发语言：Java。
- 开发平台工具：MyEclipse6.0。
- 应用服务器:Tomcat6.0。
- 数据库：SQL Server2000。
- 服务器操作系统：Windows 2003 Server。
- 客户端：Windows 2000/WinXp 和 IE 浏览器。

2．硬件环境

要求该系统能沿用现用 OA 平台硬件。

前面是对该系统整体功能需求的一个详细说明，后边将以该项目的系统管理模块为例讲解本书知识点在该项目中的具体应用。

习　题

1．该系统有几个模块？每个模块又有哪些子模块？

2．在系统设计中，首先需要设计哪一个模块？

3．本系统所采用的开发平台是什么？数据库是什么？能否采用其他数据库进行开发？

第 4 章　HTML

📖 知识点

- HTML 概述
- HTML 的基本格式
- HTML 的常用标记
- HTML 的表单设计
- HTML 的表格设计

📣 难点

- HTML 的表格设计

✍ 掌握

- HTML 的常用标记的使用
- 学会编写 HTML 的网页文件

任务引入

在 Java Web 项目开发中，通常采用分层架构设计，即视图层+业务逻辑层+数据持久层。在 3 层架构中首先需要掌握的是 Web 的视图层。而视图层通常使用 HTML 和 JSP 表示，并且 JSP 的基础是 HTML，因此有必要掌握 HTML 的用法。本章首先讲解 HTML 的基本概念及用法，然后再结合实际项目讲解 HTML 的具体应用。

4.1　HTML概述

HTML 是超文本标记语言，其英文全称为 Hypertext Markup Language。它是在普通文本文件的基础上，添加一系列的标记而组成的，这些标记可以描述输出网页的格式、颜色、字体等。如果还有一些图片、声音、动画或其他形式的资源，HTML 也会告诉浏览器到哪里去查找这些资源及这些资源放在网页的什么位置。所有的网页都是以 HTML 为基础再添加一些其他的语言（如 JSP，JavaScript，ASP 等）而构成的。HTML 作为一种网页编辑语言，易学易懂，能制作出精美的网页效果。它的文件扩展名是 html 或 htm，不需要编译，直接运行就可以了。

4.1.1　HTML的基本格式

HTML 文件以<html>开头，以</html>结束。其他的标记都必须被放在<html>…</html>内。在其中通常包含两大部分，即头部和体部。头部是<head>…</head>，用来确定文件的标题和其他关于文本的信息。体部是<body>…</body>，用来确定网页的样式。HTML 的基本格式如下。

```
<html>
<head>
```

```
<title>文件的标题</title>
</head>
<body>
    文本内容
</body>
</html>
```

一般的 HTML 标记都有起始标记和结束标记，它们成对出现。HTML 不区分大小写，<HTML>和<html>是一样的。

4.1.2　HTML的常用标记

1．<body>

<body>…</body>是 HTML 的主体部分，在其中可以添加其他的标记，其所定义的内容会在网页中如实地显示。在这个标记中可以带各种属性。例如，<body bgcolor="blue">设置背景颜色为蓝色；<body text="#ff0000">设置文本颜色为红色；<body background= "qqq.gif">设置用于填充网页的背景图文件 qqq.gif。

2．<!--注释的内容-->

在 HTML 中可以添加注释，注释以<!--开始，以-->结束。浏览器显示 HTML 网页时，不显示注释内容。

3． 、<pre>和

在 HTML 中不显示换行和空格，那么要表示段落，就需要采用段落标记 。 可以单独使用，也可以与 成对出现。使用 … 可将它们之间的文本以段落的形式显示。它会产生一个空白行间距。

使用<pre>…</pre>可以保留文本中的空格和换行符。也就是说，按照所输入的样子原样输出，所见即所得。

标记是创建一个回车换行。

例 4_1：输出个人信息。

```
<html>
<head>
<title>  个人信息</title>
</head>
 <!--显示文本颜色为红色-->
<body text="red">
姓名:张飞  <br>年龄:19<br>  学号:2006010101
<p>
    专业介绍
</p>
<pre>
     我正就读于某信息技术学院计算机软件专业
```

```
学过的课程有 Java,JSP,C,DB2 等
</pre>
<p>
    特长介绍
</p>
<pre>
    我爱好打篮球、踢足球等
</pre>
</body>
</html>
```

该文件以 Ex4_1.html 作为文件名进行保存，双击运行显示如图 4.1 所示。

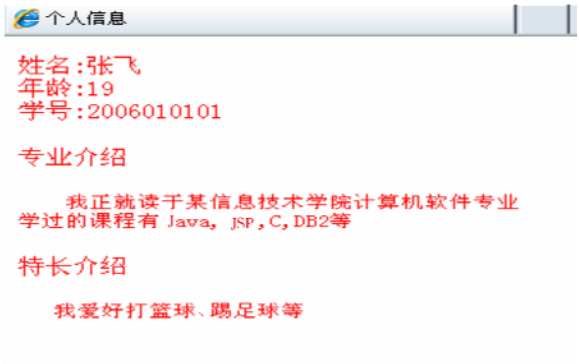


图 4.1　例 4_1 程序运行结果

4．水平线

在页面中添加水平线，可以使具有不同功能的文字分割开，达到整齐和美观的目的。在 HTML 中，利用<hr>标记使光标在此处换行，并加上一条水平线。在 HTML 中，还可以利用此标记的各种属性来对水平线进行各种设置。使用 color 属性设置水平线的颜色；使用 width 设置水平线的长度，也可以将其值设为水平线占窗口总长度的百分比；使用 size 属性设置水平线的粗细；noshade 属性不用给值，它用来加入一条没有阴影的水平线，不加入此属性，水平线将有阴影。例如，<hr color="blue"align="left"width="80%"size="1">表示画一条水平线，其颜色是蓝色，线条左对齐，画到窗口大小的 80%，粗细为 1，有阴影。

4.2　文本处理

对于网页上的文本也需要考虑格式问题。设定格式，可以让文本的输出更加美观。常见的文本格式处理有如下几方面。

4.2.1　标题处理

在 HTML 中一共有 6 种标题，分别为<h1>…</h1>到<h6>…</h6>，代表标题 1 到标题 6。标题 1 是最大号标题。标题的显示需要考虑对齐方式，对齐方式共有 3 种：left（左对

齐）、center（居中对齐）和 right（右对齐）。对齐方式可以通过设置标题的属性 align 得到。其格式如下。

```
<hn align="对齐方式">文字</hn>
```

例如，本节的标题可以被描述为：<h2 align="center">4.2 文本处理</h2>。

4.2.2　字体设置

在网页中文字可以采用不同的大小、字体、字形和颜色。这都是通过…来实现的。其格式如下。

```
<font size="字体大小" face="字体名" color="字体颜色">文字</font>
```

size 属性用来设置字体的大小。属性值为 1～7 的数字。7 号字最大，1 号字最小。默认为 3 号字。

face 属性用来设置字体。常用的英文字体有“Times New Roman”和“Arial”等，常用的中文字体有“宋体”、“隶书” 、“楷体_gb2312”、“黑体”等。用户可以指定一种或多种字体。例如，文本处理。默认的中文字体为“宋体”，默认的英文字体为“Times New Roman”。

color 属性用来设置字体的颜色。属性值可以是颜色名称或十六进制数。例如，文本处理。

文字的显示还可以有黑体、斜体、添加下画线的形式。可以通过…设置为黑体，通过…设置为斜体，<u>...</u>用来给文本添加下画线。

文字居中显可以使用 … 实现。

例 4_2：输出一首诗。

```
<html>
<head>
<tttle> New Document </tttle>
</head>
<body>
   <h1 align="center">登鹳雀楼</h1>
   <hr color="#ff0000" width="80%" size="2">
   <h4 align="center">
   <font size="4" color="blue" face="隶书" >
      <b>白日依山尽<br>
          黄河入海流<br> </b>
      <i>欲穷千里目<br>
          更上一层楼 br></i>
   </h4>
</body>
</html>
```

4.2.3　列表

文字采用列表形式，可以使文本看起来整齐、美观、层次分明和一目了然。在 HTML 中可以创建两种形式的列表。第一种是使用<dl>…</dl>创建一个普通的列表，使用<dt>…</dt>创建列表中的上层项目，使用<dd>…</dd>创建列表中的下层项目。需要注意的是<dt>…</dt>和<dd>…</dd>必须被放在<dl>…</dl>内。第二种是使用
…
创建一个标有数字的列表，使用
…
创建一个标有圆点的列表，使用
…
创建一个列表项。需要注意的是
…
只能在
…
或
…
内使用。若
…
被放在
…
内则每个列表项会加上一个数字，若被放在
…
内则每个列表项会加上一个圆点。可以设置它的 type 属性，来对圆点进行设定。type 的属性值可以是 disc（实心圆点）或 square（实心方框），默认是 disc。

例 4_3：采用列表输出学生信息。

```
<html>
<head>
<title>创建列表</title>
</head>
<body>
  <H1 align="center">学生信息一览表</H1>
  <font color="blue" size="5">
    <dl>
      <dt><u>计信一班</u></dt>
        <dd> 王新 男 1988 年出生</dd>
        <dd> 张林 女 1987 年出生 </dd>
      <p align="left">
      <u>计信二班</u>
      <ul>
        <li> 张美 女 1988 年出生</li>
        <li> 刘新 男 1987 年出生 </li>
      </ul>
      <u>计信三班</u>
      <ol>
        <li> 王汉 男 1988 年出生</li>
        <li> 孙兰 女 1987 年出生 </li>
      </ol>
    </dl>
  </font>
</body>
</html>
```

4.2.4　特殊符号

如果用户需要在网页上显示一些特殊符号，如空格、人民币符号“￥”和“£”等及与 HTML 语法相冲突的符号（如“<”和“>”），就需要使用参考字符来表示，而不能直接输入。参考字符以“&”开始，以“;”结束。常见的参考字符有： ；表示空格，&lt 表示小于号“<”，&gt 表示大于号“>”，&yen 表示“￥”等。

4.3　超链接处理

在 HTML 中建立超链接使用的是<a>…。建立超链接必须指出超链接的目标，这通过该标记的 href 属性来实现。使用<a>...标记可以链接到任何对象，它可以是一个网页，也可以是一张图片或一段文字。

最常见的超链接就是指向其他网页的超链接，当用户单击这个链接时就可以跳转到相应的网页。href 属性的值是网址或相对路径。例如：

```
<a  href= "http://www.sina.com.cn">新浪网站</a>
```

使用超链接时，一定要确保 href 属性所指的网页存在于所指定的位置，否则会无法显示该页面。默认的超链接颜色为蓝色并且有下画线。当用户将鼠标移到该链接时鼠标会变成手的形状。可以用<body>标记的 link 属性设置超链接颜色的显示。

除了链接到其他网页，也可以链接到同一网页的其他地方，也就是锚点链接。如果 HTML 文件很长，从头到尾浏览网页很费时间，这时用锚点链接就会很方便。要建立锚点链接，首先需要建立锚点，即建立一个记号，以便随后可以返回到该地方。

建立锚点使用标记：

```
<a name="锚点名字">此处创建一个锚点</a>
```

创建锚点就是为了在 HTML 文件中实现页内跳转，跳转到锚点所在的地方。跳转到锚点所在地使用标记：

```
<a href= "# 锚点名">单击此处跳转到锚点处</a>
```

如果 href 的值指定为一个单独的“#”号，表示是空链接，不进行任何跳转。

电子邮件链接也是经常使用的一种链接方式。电子邮件链接是把 href 的属性值设定为 mailto:邮箱地址。例如：

```
<a href="mailto:sunlugogo@163.com">作者的邮箱</a>
```

当浏览网页的用户单击了该超链接“作者的邮箱”，系统会自动启动邮件客户程序，默认为Outlook Express，并在邮件地址栏中填写“sunlugogo@163.com”。用户可以编辑并发送邮件。

例 4_4：超链接的使用。

```
<html>
<head>
<title> 书籍信息 </title>
</head>
<body>
    <p align="center">基于 Java 技术的 Web 应用开发</p>
     <a href="#mao1">适用范围</a>
     <a href="#mao2">特点</a>
     <a href="mailto:sunlugogo@163.com">作者邮箱</a>
     <a href="http://www.163.com">链接到 163 网页</a>
     <p><a name="mao1"></a><h3>适用范围</h3>
     <p>该书为高职教材，适用于本科、专科。同时也可作为初学者的参考书籍。</p>
<p><a name="mao2"></a><h3>特点</h3>
<p>本书语言浅显易懂，对进行 Web 开发的各个环节都予以介绍，并用一个项目贯穿本书的各个章节。理论知识不求全面，以必需、够用为度。重点在于实用技术。</p>
</body>
</html>
```

程序运行时，为了能反映出锚点的链接效果，可以缩小浏览器窗口，使之只能出现“适用范围　特点　作者邮箱　链接到 163 网页”一行。

4.4　图像处理

4.4.1　图像的插入

为了使网页看起来美观，可以在网页中插入图像。可以在网页上使用的图像有 GIF 格式（扩展名为 gif）、JPEG 格式（扩展名为 jpg）和 PNG 格式（扩展名为 png）和矢量格式（扩展名为 swf）等。在 HTML 中，采用<img>标记加入图像。该标记有两个基本属性：使用 src 属性给出要插入图像的文件名，必须包含绝对路径或相对路径；使用 alt 属性设置当鼠标移动到图像上时显示的文本，或者设置不能显示图像时显示的文本或能显示图像但显示时间过长时先显示的文本。插入图像的格式如下。

```
<img src="文件名" alt＝"文本">
```

<img src="picture.gif" alt="这是一幅美丽的风景画！"> 表示插入文件名为 picture.gif 的图像，当鼠标移动到该图像上时，显示“这是一幅美丽的风景画!”。

注意：src 必须要有值。

4.4.2　设置图像的属性

1．设置图像的大小

在 HTML 中，可以通过 width 和 height 属性指定图像的宽和高，以此来设置图像的大小。但它们的单位默认是像素，并不是常用的厘米。width 和 height 的取值既可以是像素值

也可以是百分数。如果用百分数，表示图像占当前浏览器窗口大小的比例。

2．设置图像的边框

使用 border 属性可以给图像添加边框。可以取大于或等于 0 的整数，默认单位为像素。如果取值为 0，表示图像不带边框。

<img src="picture.jpg" width="100" height="200" border="0">表示插入文件名为 picture.gif 的图像，该图像宽 100 像素，高 200 像素，不带边框。

3．设置图像的对齐方式

使用 align 属性可以设置图像相对于网页的水平对齐方式，可以取值 left（左对齐，即图像在左面，文本在图像的右侧）、center（居中对齐）和 right（右对齐，即图像在右面，文本在图像的左侧）；使用 align 属性还可以设置图像相对于周围文字的对齐方式，可以取值 top（顶部对齐）、middle（居中对齐）和 bottom（底部对齐），默认为底部对齐。例如，<img src="picture.gif"align="top"> 表示图像与文本顶部对齐。

4．设置图像的边距

可以通过 hspace 和 vspace 属性设置图像的边距，也就是设置图像周围的空白区域。使用 hspace 属性设置图像的左右边距，即图像水平方向的空白；使用 vspace 属性设置图像的上下边距，即图像垂直方向的空白；默认单位为像素。

例 4_5：插入图像。

```
<html>
<head>
<title> 插入图像</title>
</head>
<body>
    <p><img src="fly.jpg" alt="飞翔" width="70%" height="70%" border="4"
            align="center" hspace="50" vspace="50">这是一幅多美的图画呀！<p>
</body>
</html>
```

4.5　表单设计

通常在上网时，不仅需要浏览阅读网页，还需要把自己输入的信息发送给服务器，由服务器进行处理后，再把处理结果传送回客户端的浏览器上，这就是网页的交互性。使用 HTML 中的表单就可以设计允许用户和服务器进行交互的页面。在表单中可以放置表格、图像、文字、文本框、密码框、复选按钮、单选按钮、列表框等。

4.5.1　表单的基本结构

```
<form name="表单的名字" method="提交的方式" action="处理该表单的网页">
  <input type="控件" name= "" >
```

```
</form>
```

表单以<form>开始，以</form>结束。它们中间的内容都属于表单的内容。<form>标记可以带各种属性。使用 name 属性指出表单的名字，如果该网页中有多个表单，该属性必不可少，如果该网页中只有一个表单，name 属性可以省略。使用 method 属性指出提交表单的方法，可以是 get 或 post，如果以 get 方法提交，用户输入的内容会在地址栏中显示，并且速度较慢，传送的容量较小；而 post 方法刚好相反，用户输入的内容不会在地址栏中显示，速度比 get 方法快，传送的容量也比 get 方法大得多。如果用户输入密码等保密性的信息，就需要以 post 方法提交。使用 action 属性给出处理该表单的网页。例如：

```
<form action="doJsp.jsp" name="frm1" method="post">
```

该表单名字为 frm1，以 post 方法提交给 doJsp.jsp 页面进行处理。

<input>标记用来定义一个用户的输入区，使用 type 属性给出表单控件的类型，使用 name 属性指出该控件的名字。

4.5.2　文本框和密码框

文本框的类型名为 text，是一个单行文本，用户可以在文本框中输入姓名、年龄等输入内容不多的信息，输入的信息都可以看到。密码框，顾名思义，输入的内容以特殊字符代替，用户不能看到输入的信息，其类型名为 password，其格式如下。

```
<input type="text" name="" size="">
<input type="password" name="" value="" size="" maxlength="">
```

4.6　表格设计

在 HTML 中，一个表格由表、表头、行和单元格 4 部分组成。它们需要用不同的标记来完成。

1．<table>标记

<table>标记用来创建表格。表格中的所有内容及所有标记都应该位于<table>…</table>内。其格式如下。

```
<table bgcolor= "" border= "" bordercolor= "" bordercolorlight= "" bordercolordard= "" cellspacing= "" cellpadding= ""  width= "">
</table>
```

<table>标记中的各种属性可以设置或不设置。使用 bgcolor 属性设置表格的背景颜色；使用 border 属性设置边框的宽度，默认为 0；使用 bordercolor 属性设置边框的颜色；使用 bordercolorlight 属性和 bordercolordard 属性分别设置边框的明亮部分和背光部分的颜色，这两个属性只有在 border 属性值大于或等于 1 时才起作用；使用 cellspacing 属性设置表格的各个单元格之间空间的大小；使用 cellpadding 属性设置表格单元格边框与其内部内容之间空间的大小；使用 width 属性设置表格的宽度，取值为绝对像素值或总宽度的百分比。例如：

```
<table border= "1" bordercolor= "red" cellspacing= "2" cellpadding="2" width= "80%" >
</table>
```

2．<TH>标记

<th>标记用来设置表头。该标记可以为空，</th>可以省略。例如：

```
<th width=2 align="center">表格处理</th>
```

3．<caption>标记

<caption>标记紧跟在<table>标记之后，用来设置表格的标题。例如：

```
<caption align="center">表格处理</caption>
```

4．<TR>标记

<tr>标记用来定义表格的每一行。该标记只能被放在<table>…</table>内，</tr>可以省略。

5．<TD>标记

<td>标记用来创建表格中的每个单元格。该标记只能被放在<tr>…</tr>内，就是先创建行，再创建每一个单元格。例如：

```
<td align="center" width="10" colspan="3" rowspan="4" nowrap="true"></td>
```

上例表示创建单元格，内容居中对齐，单元格的宽度为 10，单元格跨占的列数为 3，单元格跨占的行数为 4，单元格中的内容不能自动断行。

4.7　HTML在项目中的应用

前面几节讲解了 HTML 本处理、表单设计及表格设计等重要内容。现在以人力资源项目中的登录页面为例讲解 HTML 的用法。登录页面如图 4.2 所示。

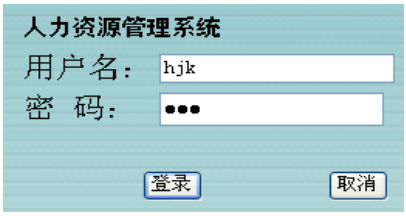


图 4.2　人力资源管理系统登录页面

其主要源代码如下。

```
A.<html>
    <head>
        <title>人力资源管理系统登录页面  </title>
}

        B.<script language="javascript">
```

```
function check(){
                    var account=document.login.useraccount.value;
                    var pwd=document.login.userpwd.value;
                    if(account==""){
                        alert("请输入账号!");
                        document.login.useraccount.focus();
                        return false;
                    }
                            if(pwd==""){
                        alert("请输入密码!");
                        document.login.userpwd.focus();
                        return false;
                    }
                    document.login.action="../../bank/UserLoginAction";
            }
        </script>
        </head>
C.<body>
<div>
D. <form name="login" method="post">
<table >
                        <tr>
                            <td align="right">
                             <font face= "黑体" size="1">人力资源管理系统</font>
                            </td>
                        </tr>
                          </table>
                <font color="red"> <%
        String state = (String) session.getValue("state");
        if (state == null)
            state = "";
 %> <%=state%> </font>
                    E.<table>
                            F.<tr>
                                G.<td>
                                    用户名:
                                </td>
                                <td>
                                    H.<input type="text" name="useraccount">
                                </td>
                            </tr>
                            <tr>
                                <td>
                                    密 码:
```

```
                                    </td>
                                    <td>
                                        <input type="password" name="userpwd">
                                    </td>
                                </tr>
                            </table>
                            <br>
                            <table>
                                <tr>
                                    <td>
                                              
                    I.<input type="submit" value="登录" onclick="return check()">
                                    </td>
                                    <td>
                                              
                                        J.<input type="reset" value="取消">
                                    </td>
                                </tr>
                            </table>
                        </form>
                    </div>
                </body>
            </html>
```

现在对上边代码进行详细分析。在 A 标记处是 HTML 的开始标记<html>，最后以</html>结束。其他的标记都必须被放在<html>…</html>内。在其中通常包含两大部分，即头部和体部。头部是<head>…</head>，用来确定文件的标题和其他关于文本的信息，如<head> <title>人力资源管理系统登录页面 </title>…</head>。体部是<body>…</body>间，用来确定网页的样式。如代码中 C 标记处所示是体部的开始部分。在 B 标记处是一段 JavaScript 代码，主要用来验证输入，若输入有错误或未输入则将弹出提示信息。在该页面中，主要是验证输入的用户名和密码是否为空，若为空，则弹出相应提示。如若没有输入用户名而进行了提交，则会弹出窗体内容为“请输入账号!”以提示用户输入账号，只有当用户输入完账号后方可正常提交。密码验证也是如此。在这里不讲解 JavaScript 的具体用法，在第 6 章会讲解其概念及用法。D 标记处是表单开始处，其中包含了两个属性。一个是 name 属性，说明该表单的名字，便于 JavaScript 验证使用；另一个属性是 method 属性，指出提交该表单的方法。E 标记处是表格开始处，在<table>…</table>内包含了<tr>标记（F 标记处），在<tr>...</tr>内包含了<td>标记（G 标记处）。H 标记处表示文本的输入。在<input>标记中有 type 属性，值为 text，表示该输入为文本内容，若 type 值为 password，表示该输入为密码，内容不可见。<input>标记中的 name 属性，应该是文本输入不可缺少的一个属性，该属性表示输入控件的名字，在提交表单数据时后台 action 根据 name 属性所对应的值而获取输入值。I 标记处表示输入类型为提交，type 值为 submit，表示该按钮具有提交表单数据功能，value 属性表示该按钮的名字，onclick 属性表示单击“登录”按钮后将调用 JavaScript 中的 check()方法，用于提交前客户端表单数据验证。若输入按钮 type 值为 reset，则表示将表单输入框数据自动清空，便于重新输入新值，如源代码 J 处所示。

4.8　实训操作

在此次实训中，主要是要熟练掌握 HTML 的基本用法，如表单的用法、表格的用法、图像处理等。

现在结合人力资源管理系统的用户管理模块，用 HTML 设计出如图 4.3 和图 4.4 所示的功能。通过该模块超级管理员可以设置一般管理员的权限，管理该系统的后台信息。功能包括：①添加新用户；②为用户设置姓名、性别、登录账号、登录密码、确认密码；③删除用户；④密码修改，单击“编辑”按钮可以进行密码修改；⑤分配角色，0 为超级管理员，1 为普通管理员。

（1）用户管理页面如图 4.3 所示。

图 4.3　用户管理页面

（2）当单击“添加用户信息”按钮时，弹出新增用户信息页面，如图 4.4 所示。

图 4.4　新增用户信息

习　题

一、单项选择题

1．下面哪一个标记可以被用来建立超链接？（　　）

A．<b>　　C．<r>

B．<a>　　D．<c>

2．下面哪一个标记是换行标记？（　　）

A．
　　C．

B．<enter>　　D．

3．下面哪一个标记是文字居中的标记？（　　）

A．　　C．<c>

B．　　D．<b>

4．下面哪一个标记可以被用来显示图形？（　　）

A．<img>　　C．<bmp>

B．<jpg>　　D．<a>

5．下面哪一个标记可以被用来显示一条水平线？（　　）

A．<line>　　C．<h1>

B．<hr>　　D．<bar>

6．显示标题采用下面哪一组标记？（　　）

A．<title>…</title>　　C． …

B．<body>…</body>　　D．<face>…</face>

7．下面哪些不是标题显示的对齐方式？（　　）

A．left　　C．middle

B．center　　D．right

8．<font>标记的哪一个属性可以被用来设置字体?（　　）

A．size　　C．face

B．color　　D．left

9．给字体加下画线采用哪一个标记？（　　）

A．<b>　　C．<u>

B．<i>　　D．<l>

10．密码框所采用的类型为（　　）。

A．text　　C．pass

B．password　　D．form

11．创建表格采用下面哪一个标记？（　　）

A．< table >　　C．< form >

B．< title >　　D．< body >

二、编程题

编写程序，制作一份个人简历，要求包含以下几项内容。

基本信息（采用标题形式）。

采用表格形式输出个人的基本信息：姓名、性别、年龄，政治面貌、出生日期。

个人特长（采用标题形式）。

字体采用宋体，字号为 5 号字，粗体，斜体。

上学经历（采用标题形式）。

采用表格的形式，字体采用宋体。

第 5 章　Servlet 技术

📖 **知识点**

- Servlet 的有关概念和特点
- Servlet 的生命周期
- Servlet 的 API
- Servlet 对表单的处理
- 使用 Cookie
- 在 Servlet 中调用其他文件

📣 **难点**

- Servlet 对表单的处理
- Servlet 的 API

✍ **掌握**

- Servlet 的 API
- Servlet 对表单的处理
- 使用 Cookie
- 在 Servlet 中调用 JSP

任务引入：

在 Web 项目开发中，如何将视图层即 HTML 或 JSP 页面的数据传到服务器端，服务器端接收数据后又如何进行处理，最后如何将处理的结果返回给客户端视图层，这就是本章需要解决的问题。为了解决上述问题，Sun 公司推出了 Servlet 技术。本章就讲述 Servlet 的概述、Servlet 的处理流程、Servlet 的生命周期及所对应的 API 的使用等重要概念。然后再结合实际项目讲解 Servlet 在项目中的具体应用。

5.1　Servlet 简介

5.1.1　Servlet 概述

Servlet 是 Sun 公司于 1997 年 3 月推出的基于 Java 的动态网页技术。Servlet 和 Applet 非常相似，Applet 是运行在客户端的 Java 小应用程序，而 Servlet 是运行在服务器端的 Java 程序，可以动态生成 Web 页面，所以 Servlet 不受客户端的限制，能够为用户提供非常广泛的功能。Servlet 可以解析 HTML 表单数据，读取和设置 HTTP 表头，处理 Cookie，跟踪会话状态等。Servlet 还可以与其他系统资源交互，例如，它可以调用系统中其他的文件，访问数据库、Applet 和 Java 应用程序等，以此生成返回给客户端的响应内容。由于 Java Servlet API 在 Web 服务器和 Servlet 之间定义了一个标准的接口，并且 Servlet 是用 Java 编

写的，这使得 Servlet 能够跨平台和不同的 Web 服务器交互。几乎所有的主流服务器都直接或通过插件支持 Servlet。Servlet 支持多线程，所以 Servlet 只要被 Web 服务器装载一次，就能被每一个客户端请求调用。这就意味着，Servlet 只在第一次执行时才需要加载，随后的请求不需要再次加载，这提高了服务器的效率。并且 Servlet 能维持请求之间的系统资源，如数据库连接。Servlet 不会因每一个请求而产生一个多余的新的 Servlet 实例。Servlet API 与协议也是无关的。Servlet 能用于多种协议，如 HTTP 和 FTP。这里主要介绍 HttpServlet。

5.1.2　Servlet的处理流程

为了能处理客户端和服务器之间的通信，Servlet 实现了一个通用的请求响应范例。Servlet 的处理流程如图 5.1 所示。其步骤如下。

（1）客户端发送一个请求给服务器端。

（2）服务器将请求信息发给 Servlet。

（3）Servlet 引擎，也就是 Web Container，会调用 Servlet 的 service()方法。

（4）Servlet 建立一个响应，并将其传递给服务器。这个响应是动态建立的，相应的内容通常取决于客户端的请求，在这个过程中也可以使用外部资源。

（5）服务器将响应返回给客户端。

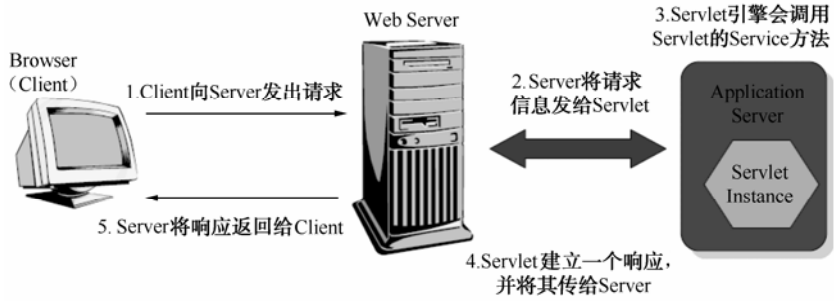


图 5.1　Servlet 的处理流程

5.1.3　Servlet的基本结构

例 5_1：输出“Welcome study Servlet!”。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Ex6_1 extends HttpServlet
{
protected void doGet(HttpServletRequest request,HttpServletResponse response)throws
ServletException,IOException
{
    PrintWriter out=response.getWriter();
```

```
        out.println("Welcome study Servlet!");
    }
    }
```

程序的运行结果如图 5.2 所示。

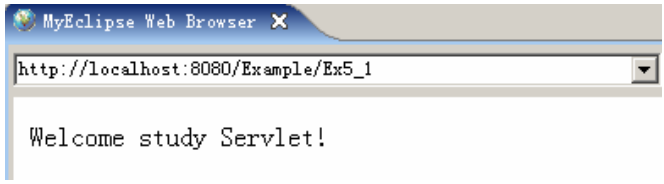


图 5.2　例 5_1 程序运行结果

上述代码是一个生成纯文本的简单 Servlet。从这个简单的例子可以看到，所有的 Servlet 都必须继承 javax.servlet.Servlet 接口。虽然可以通过直接继承该接口开发 Servlet 程序，但因为大多数 Servlet 是针对用 HTTP 协议的 Web 服务器的，所以最常用的开发 Servlet 的方法是使用 javax.servlet.http.HttpServlet 类。HttpServlet 类通过扩展（extends）GenericServlet 基类继承 Servlet 接口，提供了处理 HTTP 协议的功能。它的 service()方法支持标准 HTTP1.1 请求。

例 5_1 描述了一个基本的 Servlet 的轮廓，这个 Servlet 处理 get 请求。get 请求是 Web 页面的一种常用的浏览器请求类型。当用户在地址栏里敲入地址访问一个 Web 地址，或者提交一个未指定 method 的 HTML 表单或指定 method 为 get 的 HTML 表单时，浏览器将产生这种请求。Servlet 也可以处理 post 请求，即当 HTML 表单的 method 被指定为 post 时。

Servlet 不像普通的应用程序那样有 main()方法，它通过一些特殊的方法启动、执行和退出。service()方法用于执行，是 Servlet 程序默认的入口点。每当客户端请求一个 HttpServlet 对象时，该对象的 service()方法就要被调用，而且传递给这个方法一个请求对象 HttpServletRequest 和一个响应对象 HttpServletResponse 作为参数。在 HttpServlet 中已存在 service()方法，默认的服务功能是调用与 HTTP 请求的方法相应的 do 功能。如果 HTTP 请求方法为 get，则在默认情况下就调用 doGet()方法；如果 HTTP 请求方法为 post，则调用 doPost()方法。因此在编写 Servlet 程序时，不需要重载 service()方法，只需要根据需要调用相应的 do 方法即可。

doGet()方法或 doPost()方法都必须取决于两个参数，分别是 HttpServletRequest 对象和 HttpServletResponse 对象。通过 HttpServletRequest 对象的方法可以得到输入信息，如表单数据、HTTP 请求头及客户端的主机名等。HttpServletResponse 对象的方法指定输出信息，如 HTTP 状态代码和相应头。并且，它能够获得 PrintWriter，利用 PringWriter 可以将文档内容发给客户端。

注意：Servlet 程序必须导入所需的 package。程序中 3 个 import 语句是其他 Java 包的导入声明。导入 java.io 包，在程序中就可以访问一些标准 IO 类，如 PrintWrite。javax.servlet 和 javax.servlet.http 的导入是非常重要的，这样才能在程序中访问 Java Servlet API 的一系列类和接口的方法。

5.1.4　生成HTML的Servlet

例 5_1 的程序是生成纯文本的 Servlet，其实大多数的 Servlet 是生成 HTML，以网页的格式输出，而不是生成纯文本的的 Servlet。为了输出 HTML，需要两个额外的步骤。第 1 个步骤是通过 response.serContentType("text/html");语句设置 HTTP 响应头信息内容类型为 HTML，第 2 个步骤是修改 println 语句以建立合法的 HTML。

例 5_2：生成 HTML 的 Servlet，输出“Welcome study Servlet!”。

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Ex5_2 extends HttpServlet
{
protected void doGet(HttpServletRequest request,HttpServletResponse response)throws ServletException,IOException
{
    response.set Content Type("text/html");
    PrintWriter out=response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>这是一个生成 HTML 的 Servlet！ </title>");
    out.println("<body bgcolor=yellow>");
    out.println("<h1>Welcome study Servlet!</h1>");
    out.println("</body>");
    out.println("</html>");
}
}
```

程序的运行结果如图 5.3 所示。

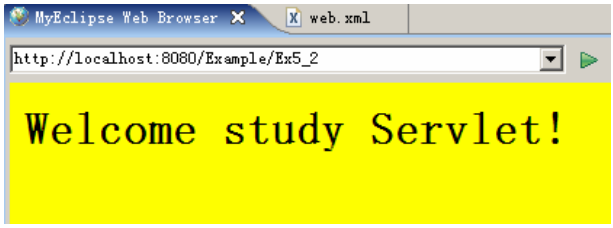


图 5.3　例 5_2 程序运行结果

这是一个简单的生成 HTML 的 Servlet 的例子，定义中类继承了 HttpServlet，因此它是基于 HTTP 协议的 Servlet。在这个类中实现了 doGet()方法。在其中对响应对象作了一些处理，响应对象将返回一个 HTML 页面给客户端。

程序首先通过 HttpServletResponse 的 setContentType()方法把响应对象的内容类型设置

成 text/html 类型。然后定义 out 为响应的 PrintWriter 对象，通过 out 的 println 语句把 HTML 文本写到响应中，送回给客户端，客户端就会显示相应的 HTML 页面。

5.1.5　Servlet 的生命周期

学习过 Java 语言的人都知道，Applet 小应用程序是运行在客户端浏览器中的 Java 小应用程序，而现在要描述的 Servlet 是运行在服务器端的 Web 容器中的 Java 小应用程序。它们有许多的相似之处，Servlet 和 Applet 都不能单独运行，都有不同于普通 Java 程序的生命周期和特别的方法。

Servlet 的生命周期定义了一个 Servlet 如何被加载和被初始化，以及它怎样接收请求、响应请求和怎样提供服务。在 Servlet 的生命周期中，会依次调用 init()、service()、destroy()方法。Servlet 的生命周期主要由 3 个过程组成。

1．初始化

如果已经配置了自动装入选项，那么在启动服务器时将自动装入 Servlet，并初始化；如果没有配置自动装入选项，那么在启动服务器后，当客户端首次向 Servlet 发出请求时，将会初始化 Servlet。装入 Servlet 后，服务器创建一个 Servlet 实例并调用 Servlet 的 init()方法。在 Servlet 的生命周期中，仅执行一次 init()方法。初始化 Servlet 时，Servlet 可以做些必要的初始化的工作，例如，从数据库里读取初始的数据，建立 JDBC 连接，或者引用其他的资源等。

Init()方法是 HttpServlet 中的方法，可以重写该方法。其定义如下。

```
public void init() throws ServletException
{
    初始化代码;
}
```

例如，读取自己配置参数的程序如下。

```
public void init()
{
String strDbUser=getInitParameter("DbUser");    //ie. getServletConfig().getInitParameter("...")
}
对应的 web.xml，需要设置<init-param>
<servlet>
        <servlet-name>Hello</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
        <init-param>
            <param-name>DbUser</param-name>
            <param-value>caoyu</param-value>
      </init-param>
            <load-on-startup>1</load-on-startup>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>Hello</servlet-name>
  <url-pattern>/servlet/Hello</url-pattern>
</servlet-mapping>
```

2．处理请求

当 Servlet 创建成功和初始化后，Servlet 就处于能响应请求的状态。每当服务器接收 Servlet 请求时，服务器就启动一个新的线程，在该线程中，Servlet 调用 service()方法响应客户端请求，也就是说，每个客户端的每次请求都导致 service()方法被调用执行，调用过程运行在不同的线程中，互不干扰。其定义如下。

```
public void service(ServletRequest request, ServletResponse response)throws IOException
{
处理请求代码;
}
或
public void service(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException
  {
处理请求代码;
}
```

在调用 service()时，service()方法检查 HTTP 的请求类型（get，post，put，delete），并根据其类型调用适当的 doGet()，doPost()，doPut()方法等。

注意：和 init()方法不同，init()方法只调用一次，而 service()方法需要调用多次。

3．销毁

Web 服务器可能需要删除一个以前装载的 servlet 实例，这时服务器会调用 service()的 destroy()方法。通常在这个方法中执行一些清除资源的操作，例如，释放数据库连接，停止后台线程，关闭文件等。其定义如下。

```
public void destroy()
{
      清除资源的代码;
}
```

注意：destroy()方法在 servlet 生命周期中也只执行一次。

例如：

```
public void destroy()
{
        try{
            if(myFileInputStream!=nulll)
myFileInputStream.close();
}catch(IOException e){ }
}
```

5.2　Servlet API

Servlet 通过 API 接口来处理客户端请求。Java Servlet API 是一组 Java 类和接口，它定义了 Web 客户端和 Web Servlet 之间的标准接口。客户端请求发给 Web 服务器，Web 服务器通过 Servlet API 接口调用 Servlet 为这个请求服务。

Java Servlet API 由两个包组成，分别是 javax.servlet 和 javax.servlet.http。javax.servlet 包中所包含的是编写 Servlet 所需的最基本的类和接口，这些类是独立于协议的。javax.servlet.http 包扩展了上述基础包的功能，对 HTTP 协议提供了支持。

javax.servlet.Servlet 接口是 Servlet API 的一个抽象类。这个类定义了 Servlet 必须实现的方法，如 init()方法、service()方法和 destroy()方法等。所有的 Servlet 都必须实现这个接口。GenericServlet 类已经实现了此接口，它定义了一个通用的与协议无关的 Servlet。而 HttpServlet 类继承了 GenericServlet 类，它支持 HTTP 协议。编写 Servlet 时，通常不需要直接实现 Servlet 接口，而是继承 GenericServlet 类或 HttpServlet 类。

5.2.1　通过继承GenericServlet类

GenericServlet 类定义了一个通用的与协议无关的 Servlet。GenericServlet 类提供了简单的生命周期函数 init()方法和 destroy()方法。由于 service()方法在 GenericServlet 中被声明成了抽象方法，并没有实现，所以如果要编写一个普通的 Servlet，只需要重写其中的 service()方法。

例 5_3：编写程序，在页面中输出“您好!欢迎学习 Servlet”。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Ex5_3 extends GenericServlet
{
    public void service(ServletRequest request,ServletResponse response)
throws IOException
    {
        response.setContentType("text/html;charset=GB2312");
        PrintWriter out=response.getWriter();//获得发送数据的输出流
        out.println("<html>");
        out.println("<head><title>继承自 GenericServlet 的 Servlet</title></head>");
        out.println("<body>");
        out.println("<hr width=80% align=left color=blue>");
        out.println("<br><br><font size=8 face= 隶 书 color=red> 您 好 ! 欢 迎 学 习
Servlet</font>");
        out.println("<br><br><hr width=80% align=left color=blue>");
        out.println("</body>");
```

```
            out.println("</html>");
        }
    }
```

程序的运行结果如图 5.4 所示。

图 5.4　例 5_3 程序运行结果

例 5_4：编写程序，实现一个计数器。

分析：在 Servlet 被加载之后，当后续的客户端请求 Servlet 时，服务器将启动一个线程，在该线程中，Servlet 调用 service()方法响应客户端请求，而且在 Servlet 类中定义的成员变量，被所有客户端的所有线程共享。程序如下。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Ex5_4 extends GenericServlet
{
    int counter;
    public void service(ServletRequest request,ServletResponse response)
throws IOException
    {
        response.setContentType("text/html;charset=GB2312");
        PrintWriter out=response.getWriter();
        out.println("<html><body>");
        out.println("<head><title>Servlet 中共享变量</title></head>");
        counter=counter+1;
        out.println("您是访问 Servlet 的第"+counter+"个人！ ");
        out.println("</body></html>");
    }
}
```

程序的运行结果如图 5.5 所示。

图 5.5　例 5_4 程序运行结果

5.2.2　通过继承 HttpServlet类

由于 HttpServlet 类继承了 GenericServlet 类，而 HttpServlet 类已经实现了 service()方法，service()方法会根据请求的类型是 get 还是 post 来调用 doGet()或 doPost()方法，所以在继承 HttpServlet 类时，通常在 doGet()或 doPost()方法中定义 Servlet 的功能。其程序代码如下。

```
String method = request.getMethod();
if(method.equals("GET"))
{
.....doGet(request,response);.....
}
else if (method.equals("POST"))
{
      doPost(request,response);
}
else if...../// doHead doPut doDelete doOptions doTrace
```

例 5_5：编写程序，输入一个数，求这个数的平方。

分析：需要两个文件，一个是 HTML 文件，一个是 Servlet 文件。在 HTML 文件中，创建一个表单，表单中有一个文本框（让用户输入一个数）和一个“提交”按钮，提交的方法可以是 get 或 post 方法。Servlet 文件用来处理提交的数据，进行求平方运算，并输出。

Ex5_5.html

```
<html>
<body bgcolor="cyan" >
<font color="" size="8">
<form action="Ex5_5" method="get">
      <input type="text" name="numTxt">
       <input type="submit" value="提交">
</form>
</body>
</html>
```

Ex5_5.java

```
import javax.servlet.*;
import javax.servlet.http.*;
```

```
import java.io.*;
public class Ex5_5 extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException
    {
        response.setContentType("text/html;charset=GB2312");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<head><title>继承自 HttpServlet 的 Servlet</title></head>");
        out.println("<body>");
        out.println("<hr width=80% align=left color=blue>");
        String num=request.getParameter("numTxt");
        double n=0;
        try
        {
            n=Double.parseDouble(num);
            out.print("<br>"+n*n);
        }
        catch(NumberFormatException e)
        {
          out.print("<h1>输入错误，请重新输入！</h1>");
        }
        out.println("<br><br><hr width=80% align=left color=blue>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

程序的运行结果如图 5.6 所示。

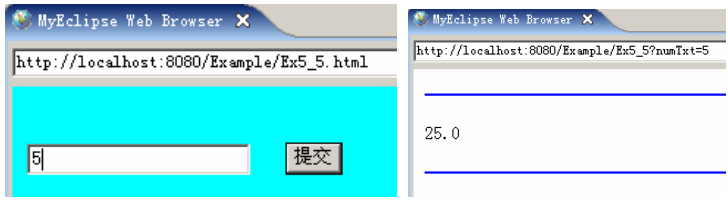


图 5.6　例 5_5 程序运行结果

注意：get 和 post 方法的区别如下。

在显示上：

如果采用 get 方法发送表单，则可以从返回页面的地址栏中看到用户输入的信息，如果采用 post 方法则不会；所以如果想要使用户输入的信息保密，则采用 post 方法。

在上传方式上：

get 方法既可以采用 URL 地址栏方式（...），也可以采用表单方式（<form [method="get"] ...>...</form>）；而 post 方法只能采用表单方式（<form [method="post"] ...>...</form>）。

在上传数据量上：

get 方法对上传数据量有限制；post 方法对上传数据量无限制，甚至可为大文件上传。

5.3　Servlet 对表单的处理

通常用 HTML 建立的网页都是静态的，这时就需要用 Servlet 对用户输入的数据进行处理，产生动态页面，即随着用户、时间或其他条件的不同返回不同的页面。当用户在表单里输入一些信息，以 get 方法提交后，会在浏览器的地址栏中看到所提交的信息。例如，在例 5_5 中，如果输入了 5，则提交后，在地址栏中可以看到“http://localhost:8080/test/Ex5_5?numTxt=5”。问号后的数据就是表单数据，它从 Web 页面获取信息，将其传送给服务器端的应用程序。

Servlet 的表单分析是自动处理的，只需要调用 HttpServletRequest 的 getParameter()方法。该方法带一个字符串类型的参数，实际上此参数为表单中控件的名字。例如，在例 5_5 中，该方法带的参数为文本框的名字。注意参数要区分大小写。用 get 方法发送数据和用 post 方法发送数据是相同的。Servlet 会判断使用的是哪种方法并自动完成参数的处理，返回值是一个字符串类型，是对应某个参数名的值。如果希望得到的是数字，则需要写代码进行类型转换。其调用格式如下。

```
request.getParameter("参数");
```

如果该参数存在，但没有值，则返回一个空串。如果该参数不存在，则返回 null。如果参数具有不止一个值，则应该调用 getParameterValues()方法，其格式如下。

```
request.getParameterValues("参数");
```

使用此方法返回多个值，为一个字符串数组。如果要一次获取多个参数，可使用 getParameterNames()方法，其格式如下。

```
request.getParameterNames();
```

使用此方法得到表单中所有参数的名字，返回一个枚举类型。

下面分别从 3 个例子来看这几个方法的用法。

例 5_6：getParameter()方法的使用。

主页面：Ex5_6.html。

```
<html>
<head>
  <title> request.getParameter()方法的使用</title>
</head>
<body>
```

```
 <form action="Ex5_6" method="post">
      <input type="text" name="t1"><br>
      <input type="text" name="t2"><br>
      <input type="text" name="t3"><br>
      <input type="submit" name="submit" value="提交">
      <input type="reset" name="reset" value="重置">
 </form>
</body>
</html>
```

处理页面：Ex5_6.java。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class Ex5_6 extends HttpServlet
{
      public void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException
      {
            response.setContentType("text/html;charset=GB2312");
            String s1=request.getParameter("t1");
            String s2=request.getParameter("t2");
            String s3=request.getParameter("t3");
            PrintWriter out=response.getWriter();
            out.println("<html>");
            out.println("<body>");
            out.println("<hr width=80% align=left color=blue>");
          out.println("您刚才输入的值为：");
          out.print("<br>"+s1+"<br>");
          out.print(s2+"<br>");
          out.print(s3+"<br>");
            out.println("<br><br><hr width=80% align=left color=blue>");
            out.println("</body>");
            out.println("</html>");
      }
}
```

程序的运行结果如图 5.7 所示。

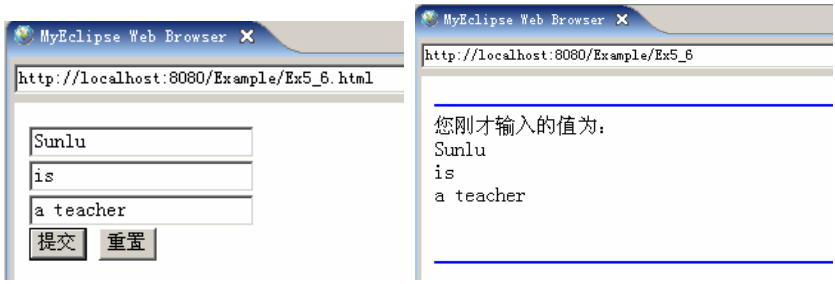


图 5.7　例 5_6 程序运行结果

在这个例子中，在 HTML 页面上定义了 3 个文本框，“提交”按钮用于提交数据，调用 Servlet 处理，以生成返回页面。返回页面将显示用户刚刚输入的个人信息。在 HTML 页面中，将用户发送的数据传送给 Servlet，是通过代码<form action="Ex5_6"> 实现的。action 的属性值为 Servlet 文件的目录。如果 HTML 文件和 Servlet 文件在同一个目录下，则不需要写目录，只需要写文件名。在 Servlet 文件中通过 getParameter()方法来获取 HTML 文件中表单中用户输入的信息。

例 5_7：getParameterValues()方法的使用。

主页面：Ex5_7.html。

```
<html>
<head>
  <title>getParameterValues()方法的使用</title>
</head>
<body>
   <form action="Ex5_7" method="post">
       请选择您所学过的专业课：<br>
      <select name="list" size="8" multiple="true">
       <option value="java">Java 程序设计
       <option value="c">C 语言程序设计
       <option value="db2">DB2 数据库
       <option value="data">数据结构
      </select>
      <input type="submit" name="submit" value="提交">
       <input type="reset" name="reset" value="重置">
   </form>
</body>
</html>
```

处理页面：Ex5_7.java。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
```

```
public class Ex5_7 extends HttpServlet
{
    public void doPost(HttpServletRequest request,HttpServletResponse response)
throws ServletException, IOException
    {
        response.setContentType("text/html;charset=GB2312");
        String s1[]=request.getParameterValues("list");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<hr width=80% align=left color=blue>");
       out.println("您刚才的选择为：<br>");
       for(int i=0;i<s1.length;i++)
       {
          if(s1[i].equals("java"))
              out.print("Java 程序设计<br>");
          else if(s1[i].equals("c"))
              out.print("C 语言程序设计<br>");
          else if(s1[i].equals("db2"))
              out.print("DB2 数据库<br>");
          else if(s1[i].equals("data"))
              out.print("数据结构<br>");
       }
        out.println("<br><br><hr width=80% align=left color=blue>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

程序的运行结果如图 5.8 所示。

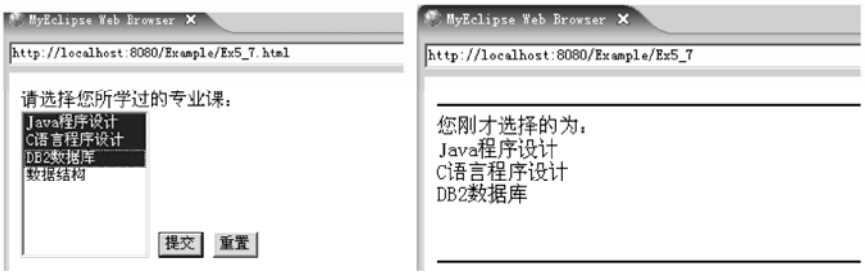


图 5.8　例 5_7 程序运行结果

在这个例子中，使用了 getParameterValues()方法。在 HTML 中，定义了一个列表框，method 属性为 post，用户输入的数据会作为 post 请求的参数进行传递。由于该列表框可以多选，形成了一个参数多个值的情况。这时候 Servlet 对其处理就必须采用

getParameterValues()方法以获取多个值。由于使用 getParameterValues()方法返回的是一个字符串数组，所以可以采用循环的方式获取每一个选中的值。需要注意的是字符串比较不能用“==”的形式，而是采用 equals()方法进行比较。

例 5_8：getParameterNames()方法的使用。

主页面：Ex5_8.html。

```
<html>
<head>
  <title>getParameterNames()方法的使用</title>
</head>
<body>
  <form action="Ex5_8" method="get">
    <p><h1></h1></p>
    姓名：<input type="text" name="sname" ><br><br>
    学号：<input type="text" name="snum" ><br><br>
    班级：<input type="text" name="sclass">
    <input type="submit" name="submit" value="提交">
    <input type="reset" name="reset" value="重置">
  </form>
</body>
</html>
```

处理页面：Ex5_8.java。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class Ex5_8 extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("text/html;charset=GB2312");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<hr width=80% align=left color=blue>");
        out.println("您刚才输入的是：");
        Enumeration e=request.getParameterNames();
        while(e.hasMoreElements())
        {
        String paraName=(String)e.nextElement();
```

```
                String paraValue=(String)request.getParameter(paraName);
                out.println("参数名称："+paraName+"<br>");
                out.println("参数内容："+paraValue+"<br>");
            }
            out.println("<br><br><hr width=80% align=left color=blue>");
            out.println("</body>");
            out.println("</html>");
        }
    }
```

程序的运行结果如图 5.9 所示。

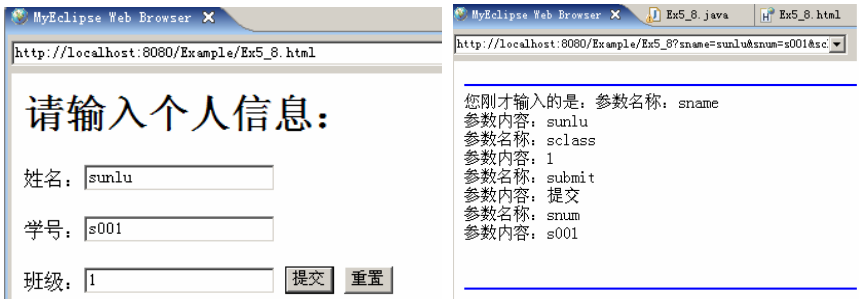


图 5.9　例 5_8 程序运行结果

在这个例子中，使用的方法是 getParameterNames()方法。在 HTML 表单中，用户输入的数据作为 get 请求的参数传递给 Web 服务器，action 属性表示当这个表单被提交时，会调用哪个 Servlet 文件来处理该表单。在这个表单中，用户输入的姓名、学号、班级、分别作为 sname 参数、snum 参数和 sclass 参数的值。可以通过 HttpServletRequest 对象访问它们。在 Servlet 中，通过 getParameterNames()方法得到所有参数的名字，也就是 sname 参数、snum 参数和 sclass 参数，然后通过 enum 的 hasMoreElements()方法，一一读出该参数，从而得到用户输入的数据。也可以通过 getParameter()方法，直接给定一个参数的名字，得到该参数的数值。

5.4　使用Cookie

HTML 是一种无状态的协议，Servlet 基于 HTTP（面向连接的无状态协议）/ Socket 80 编程，一次请求/响应（request/reponse）的过程从 socket 连接到关闭，期间是不保留任何用户的状态信息的，所以没法辨别两次的访问是否为同一用户。也就是说，Web 服务器将对某个页面的每次访问都当做相互无关的访问来处理，服务器并不保留上一次的访问信息。因此，Web 应用程序并不了解有关同一用户以前请求的信息。每当客户端检索某个页面时，它都要打开相应 Web 服务器的一个独立的连接，服务器不会自动保留客户端的状态信息。这会导致许多困难，一些商务应用必须辨别用户和保留用户的有价值的操作数据。例如，当网上商店的顾客添加一个商品到其购物车中时，服务器怎样知道购物车中已经有了哪些

商品？当顾客决定付款时，服务器怎样确定哪个购物车是他的？所以，Web 服务器需要以某种形式得到用户的状态信息。可以用 Cookie 技术达到此目的。Cookie 将用户的状态信息保存在客户端。

5.4.1　Cookie的概念

Cookie 是服务器发送给浏览器的体积很小的纯文本文件，它被存储在客户端的计算机上，用来保存用户的各种状态信息，如登录信息，这样可以使用户不用在每次登录时都重复输入这些信息。

Cookie 分为两类，持久的和暂时的。持久的 Cookie 会被存储在客户端的文件系统中。例如，笔者浏览了 yahoo 网站后，则存储该息信的 Cookie 存放于 C:\Documents and Settings\sunlu\Cookies\sunlu@www.yahoo[2].txt（yahoo 网站创建并发到我 IE 的 Cookie）。一般来说，客户端浏览器一般只允许存放 300 个 Cookie，每个站点最多存放 20 个 Cookie，每个 Cookie 的大小被限制为 4 KB，所以 Cookie 不会占用太大的硬盘空间。暂时的 Cookie 被存储在内存中，一旦浏览器关闭，Cookie 就消失了。

5.4.2　检测浏览器是否支持Cookie

由于持久的 Cookie 被存储在客户端的文件系统中，所以用户可以看到和修改 Cookie。对安全性要求比较高的信息，如密码、信用卡信息等，最好不要放到 Cookie 中，最好放到服务器的数据库中。如果需要，用户可以在浏览器端禁止使用 Cookie（方法为，打开“Internet 属性”，选择“隐私”选项卡，单击“高级”按钮进行设置）。所以，要编写关于 Cookie 的程序，用户首先必须检测浏览器是否支持 Cookie。

大多数浏览器允许用户设置其是否支持使用 Cookie。需要注意的是，并没有一种方法能够直接检测浏览器的 Cookie 设置，这时可以编写并发送一个供测试用的 Cookie，然后通过试图读取这个测试 Cookie 来判断浏览器是否支持 Cookie。

检测浏览器 Cookie 设置的步骤如下。

（1）用 add()方法发送给浏览器一个测试 Cookie。

（2）用 getCookies()方法读取测试 Cookie。

（3）根据 getCookies()的结果执行适当的代码。

5.4.3　Cookie的使用

Cookie 是一组名称数值对，每一个 Cookie 由名字和其相应的数值组成。下面介绍 Cookie 的使用方法。

1．创建Cookie

通过 Cookie 的构造方法可以创建一个 Cookie，其格式如下。

```
Cookie  名字=new Cookie(name 字串, value 字串);
```

该构造方法含有两个字符串类型参数：Cookie 的名字和 Cookie 的值。名字和值都必须符合命名规则，即由字母、数字和下画线组成。

2．将Cookie传送给浏览器

通过 response.addCookie（Cookie 对象）方法，可以将 Cookie 的数据和 HTTP 相应一起传给客户端的浏览器，就可以达到在客户端计算机上存储 Cookie 的目的。

3．设置和读取Cookie的有效期

getMaxAge()和 setMaxAge()是读取和设置 Cookie 经过多长时间就过期的两个方法。时间是以秒计，如果设置值为正数，表示该 Cookie 将存活这个正数值那么长的时间。例如，setMaxAge(2*60*60)表示设置 Cookie 的使用期限为两小时。如果没有设置值或设置值为负数，表示 Cookie 是临时的，在用户关闭浏览器时，Cookie 就会消失，并不被存储在硬盘上。设置值为 0，表示浏览器删除相应的 Cookie。

通常采用如下代码设置一个 Cookie。

```
Cookie x=new Cookie(名字 name,值 value);
x.setMaxAge(秒值);
response.addCookie(x);
```

4．读取Cookie

要从客户端浏览器读取 Cookie，需要采用 request.getCookies()方法。该方法的返回值为 Cookie 的对象数组。需要注意的是，程序不能读取某个特定的 Cookie，必须读取所有的 Cookie，循环访问该数组的各个元素，从中找出需要的那一个。

5．获取Cookie的名字和值

可以通过 Cookie 对象名.getName()方法来获取 Cookie 的名字，通过 Cookie 对象名.getValue ()方法来得到该 Cookie 的值。 这两个方法的返回值都为字符串类型。

通常采用如下代码读出已经存在的 Cookie。

```
Cookie [ ] cookies=request.getCookies();
for(int i=0;i<=cookies.length;i++)
{
    Cookie x=cookies[i];
    String name=x.getName();
    String value=x.getValue();
}
```

6．设置Cookie的值

设置 Cookie 的值采用 Cookie 对象名.setValue(值)方法。

例 5_9：编写程序显示用户上次登录访问 Servlet 的时间。

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class Ex5_9 extends HttpServlet
{
```

```
    public void service(HttpServletRequest request,HttpServletResponse response) throws
IOException
    {
        boolean firstLog=true;
        final int SECONDS=60*60*24*30;//有效期为 1 个月
        Cookie myCookie=null;
        Cookie []cookies=request.getCookies();
        response.setContentType("text/html;charset=GB2312");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<head><title>显示上次登录的时间 </title></html>");
        out.println("<body>");
        if(cookies!=null)
        {
            for(int i=0;i<cookies.length;i++)
            {
                if(cookies[i].getName().equals("PreLogin"))
                {
                    firstLog=false;
                    myCookie=cookies[i];
                    break;
                }
            }
        }
        if(!firstLog)//不是第一次登录
        {
            out.println("上次登录的时间为："+myCookie.getValue());
            Calendar calendar=Calendar.getInstance();
            Date now=calendar.getTime();
            String nowString=now.toString();
            myCookie.setValue(nowString);
            myCookie.setMaxAge(SECONDS);
            response.addCookie(myCookie);
        }
        else
        {
            out.println("你是该月第一次登录服务器，所以没有上次登录的时间记录");
            Calendar calendar=Calendar.getInstance();
            Date now=calendar.getTime();
            String nowString=now.toString();
            myCookie=new Cookie("PreLogin",nowString);
            myCookie.setMaxAge(SECONDS);
            response.addCookie(myCookie);
        }
```

```
            out.println("</body>");
            out.println("</html>");
        }
    }
```

程序的运行结果如图 5.10 所示。

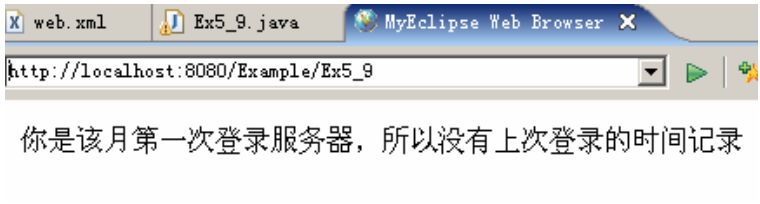


图 5.10　例 5_9 程序运行结果

5.5　Servlet实现在页面之间的跳转

在进行 B/S 架构项目的应用程序开发中，经常会用到在多个 Servlet 或页面之间进行跳转，来完成一个请求。在 Servlet 中提供了两个方法来实现跳转：forward()和 include()。

5.5.1　forward()方法

forward()方法完成转发功能，它可以把处理用户请求的功能转发给其他的 Servlet 来完成。程序被转发给另一个 Servlet 后，将不会回到原 Servlet 继续执行。该方法的使用方式如下。

```
getServletContext().getRequestDispatcher("转发的页面").forward(request,response);
```

ServletContext 是一个接口，它向 Servlet 提供访问其环境所需的方法，并记录一些重要的环境信息。通过 getServletContext()方法可以得到 ServletContext 对象。通过该对象的 getRequestDispatcher()方法，可以将一个客户端接收的请求发送给其他服务器资源的对象，包括 Servlet、HTML 或 JSP。

例 5_10：forward()方法的使用。

```
forward.html
<html>
<head>
</head>
<body>
<form action="ForwardServlet">
请选择将要查看的客户端技术:<p>
<input type="radio" name="option" value="HTML"> HTML <br>
<input type="radio" name="option" value="Java Applet" Checked> Java Applet<br>
```

```
<input type="radio" name="option" value="VBScript" >VBScript<br>
<input type="radio" name="option" value="JavaScript"> JavaScript<br>
<input type="radio" name="option" value="Activex &NetScape">Activex &NetScape<p>
<input type="submit" value="提交选择结果">
</form>
</body>
</html>

ForwardServlet.java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ForwardServlet extends GenericServlet
{
    public void service(ServletRequest request,ServletResponse response)throws
IOException,ServletException
    {
        response.setContentType("text/html;charsetGB2312");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<body>");
        RequestDispatcher dispatcher=request.getRequestDispatcher("forward.jsp");
        dispatcher.forward(request,response);
        out.println("This is from Servlet");
        out.println("</body>");
        out.println("</html>");
    }
}
forward.jsp
<PRE>

----JSP BEGIN---<P>

<%
String option=request.getParameter("option");
out.println("You have selected<B>"+option+"</B>");
out.println("displayed in JSP page");
%>

<P>----JSP END----
</PRE>
```

程序的运行结果如图 5.11 所示。

图 5.11　例 5_10 程序运行结果

5.5.2　include()方法

include()方法完成包含功能，用它可以在一个 Servlet A 中包含另一个 Servlet B，让另一个 Servlet B 完成处理用户请求的功能。和 forward()方法不同的一点在于，执行完 Servlet B 后，还会再回到 Servlet A 中继续执行。这就相当于 C 语言中的函数调用，程序在 main()执行过程中，遇到函数调用，这时程序会转去执行函数调用部分，等到执行完函数调用部分后，程序回到主函数 main()继续向下执行。该方法的使用方式如下。

```
getServletContext()..getRequestDispatcher("转发的页面").include(request,response);
```

例 5_11：include()方法的使用。

include.html

```
<html>
<head>
</head>
<body>
<form action="IncludeServlet">
请选择将要查看的客户端技术:<p>
<input type="radio" name="option" value="HTML"> HTML<br>
<input type="radio" name="option" value="Java Applet" checked> Java Applet<br>
<input type="radio" name="option" value="VBScript" >VBScript<br>
<input type="radio" name="option" value="JavaScript"> JavaScript<br>
<input type="radio" name="option" value="Activex &NetScape">Activex &NetScape<p>
<input type="submit" value="提交选择结果">
</form>
</body>
</html>
```

IncludeServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
```

```
public class IncludeServlet extends HttpServlet
{
    public void service(HttpServletRequest request,HttpServletResponse response)
throws IOException,ServletException
    {
        response.setContentType("text/html;charsetGB2312");
        PrintWriter out=response.getWriter();
        out.println("<html>");
        out.println("<body>");
        RequestDispatcher dispatcher=request.getRequestDispatcher("include.jsp");
        dispatcher.include(request,response);
        out.println("This is from Servlet");
        out.println("</body>");
        out.println("</html>");
    }
}
```

include.jsp

```
<PRE>

----JSP BEGIN---<P>

<%
String option=request.getParameter("option");
out.println("You have selected<B>"+option+"</B>");
out.println("displayed in JSP page");
%>

<P>----JSP END----
</PRE>
```

同样选择“Activex &NetScape”，则程序的运行结果如图 5.12 所示。

图 5.12　例 5_11 程序运行结果

5.6　Servlet在项目中的应用

本章主要学习 Servlet 的用法，前面已经对 Servlet 的概念及用法进行了详细的讲解。现在以人力资源项目中系统管理模块的用户组管理实现为例讲解 Servlet 在项目中的应用。

根据第 3 章人力资源管理系统需求，需要在系统管理模块中设置用户组管理。设置用户组管理的目的是便于给不同用户组里的员工分配不同的菜单及权限。在用户组管理中，需要完成添加新用户组、删除所选用户组、修改用户组、查询用户组等功能。用户组管理页面如图 5.13 所示。

图 5.13　用户组管理页面

（1）该页面原名为 groupManage.jsp,其主要源代码如下。

```
<HTML>

    <BODY>
        <!--action 的值为处理查询用户组的 Servlet 的 URL-->
        <FORM action="../../bank/GroupManageAction" method="post">
            当前位置：用户组管理
            <BR>
            用户组名：
            <INPUT type="text" name="group_name" size="20">
            用户组码：
            <INPUT type="text" name="group_id" size="20">
            <INPUT type="submit" name="sub" value="查询">
            <INPUT type="Submit" name="sub" value="查看全部">
            <BR>
            <!--location 后的值为处理该操作的 JSP 页面或 Servlet 的 URL-->
            <INPUT type="button" name="sub" value="添加新用户组"
                onclick="javascript:window.location='../../bank/UserGroup/addGroup.jsp'">
            <INPUT type="submit" name="sub" value="删除用户组">
            <BR>
            <TABLE border="1" width="892" height="345">
                <TBODY>
```

```
<%
    ResultSet rs = (ResultSet) session.getValue("session");
    GroupBean groupBean = new GroupBean();
    GroupManageDAO gmi = GroupManageDAOIMP.getFactory().getImp(
            groupBean);
    if (rs == null) {
        rs = gmi.getAllInfo();
        if (rs == null || !rs.next())
            out.print("数据中没有信息！");
        else {
%>
<TR>
    <TD colspan="2">
        用户组名
    </TD>
    <TD>
        用户组码
    </TD>
    <TD>
        用户组说明
    </TD>

    <TD align="center">
        操作
    </TD>
</TR>
<TR>
    <TD colspan="2">
        <INPUT type="checkbox" name="groupids"
            value="<%=rs.getString("group_id")%>">
        <%=rs.getString("group_name")%>
    </TD>
    <TD name="group_id">
        <%=rs.getString("group_id")%>
    </TD>
    <TD name="group_description">
        <%=rs.getString("group_description")%>
    </TD>

    <TD>
        <a
onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
```

```
>【修改基本信息】</a><a

      onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
1'">【删除】</a>
                                          </TD>
                                    </TR>
                                    <%
                                    while (rs.next()) {
                                    %>
                                    <TR>
                                          <TD colspan="2">
                                                <INPUT type="checkbox" name="groupids"
                                                      value="<%=rs.getString("group_id")%>">
                                                <%=rs.getString("group_name")%>
                                          </TD>
                                          <TD name="group_id">
                                                <%=rs.getString("group_id")%>
                                          </TD>
                                          <TD name="group_description">
                                                <%=rs.getString("group_description")%>
                                          </TD>

                                          <TD>
                                                <a

      onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
>【修改基本信息】</a>
                                                <a

      onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
1'">【删除】</a>
                                          </TD>
                                    </TR>
                                    <%
                                                }
                                                }
                                          } else if (!rs.next())
                                                out.print("没找到您需要的信息！ ");
                                          else {
                                    %>
                                    <TR>
                                          <TD colspan="2">
                                                <INPUT type="checkbox" name="userGroup">
                                                用户组名
```

```
                                    </TD>
                                    <TD>
                                        用户组码
                                    </TD>
                                    <TD>
                                        用户组说明
                                    </TD>

                                    <TD align="center">
                                        操作
                                    </TD>
                                </TR>
                                <TR>
                                    <TD colspan="2">
                                        <INPUT type="checkbox" name="groupids"
                                            value="<%=rs.getString("group_id")%>">
                                        <%=rs.getString("group_name")%>
                                    </TD>
                                    <TD name="group_id">
                                        <%=rs.getString("group_id")%>
                                    </TD>
                                    <TD name="group_description">
                                        <%=rs.getString("group_description")%>
                                    </TD>

                                    <TD>
                                        <a

    onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
>【修改基本信息】</a><a

    onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
1'">【删除】</a>
                                    </TD>
                                </TR>
                                <%
                                while (rs.next()) {
                                %>
                                <TR>
                                    <TD colspan="2">
                                        <INPUT type="checkbox" name="groupids"
                                            value="<%=rs.getString("group_id")%>">
                                        <%=rs.getString("group_name")%>
                                    </TD>
```

```
                                        <TD name="group_id">
                                            <%=rs.getString("group_id")%>
                                        </TD>
                                        <TD name="group_description">
                                            <%=rs.getString("group_description")%>
                                        </TD>

                                        <TD>
                                            <a

    onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
>【修改基本信息】</a>
                                            <a

    onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
1'">【删除】</a>
                                        </TD>
                                    </TR>
                                    <%
                                        }
                                        }
                                    %>
                                </TBODY>
                            </TABLE>
                        </FORM>
                    </BODY>
                </HTML>
```

前面用 JSP 表示视图层，在视图层中需要完成添加用户组、删除用户组、编辑用户组等功能。页面只用于显示，那么这些操作又是怎么进行处理的呢？页面输入数据被传到服务器，服务器又将数据传给 Servlet 进行处理，Servlet 处理完后又将处理结果返回给服务器，服务器又将数据返回给客户端。下面就详细讲述 Servlet 如何处理请求数据及如何响应请求数据。

（2）页面 groupManage.jsp 所对应的 Servlet 为 GroupMangeAction。其主要源代码如下。

```
package com.etop.groupmanage.servlet;
import com.etop.bank.groupmanage.business.GroupManageDAOIMP;
import com.etop.bank.groupmanage.dao.GroupManageDAO;

A．public class GroupManageAction extends HttpServlet {
    // group_id 表示用户组码
    private String group_id;

    // group_name 表示用户组名
```

```
        private String group_name;

        // group_description 表示用户组描述
        private String group_description;

        //此处的 id 为 sd_sys_usergroup 表 id 字段
        private String id;

    B．  public void doGet(HttpServletRequest request, HttpServletResponse response)throws
ServletException, IOException {
            PrintWriter out = response.getWriter();
            // 得到从页面中传过来的 sign 的值,来表示删除单一的用户组
    C．  String sign = request.getParameter("sign");
           //从页面获取 group_id 的值
            String   group_id=request.getParameter("group_id");
          //从页面获取 group_name 的值
          String group_name=request.getParameter("group_name ");
         //从页面获取 group_description 的值

      String group_description=request.getParameter("group_description");
         //从页面获取 sd_sys_usergroup 中的 id 值
            String id=request.getParameter("id");
            // groupids 数组用来存储进行多用户组删除时，每个用户组的编码
            String[] groupids = request.getParameterValues("groupids");
            //通过下面的方法来得到提交按钮 sub 的值，确定其操作类型
            String sub = request.getParameter("sub");
            GroupBean groupBean = new GroupBean();
            //使用 JavaBean 的 set 方法来对其属性变量值初始化
            groupBean.setId(id);
            groupBean.setGroupId(group_id);
            groupBean.setGroupName(group_name);
            groupBean.setGroupDescription(group_description);
            groupBean.setGroupids(groupids);
    GroupManageDAO gmi = GroupManageDAOIMP.getFactory().getImp(groupBean);
            ResultSet rs = null;
            //根据 sub 的值来执行不同的操作（类中的不同方法）
            // 根据用户填入的信息来进行条件查询
            if ("查询".equals(sub)) {
                rs = gmi.queryGroup();
                D．request.getSession().putValue("session", rs);
    E．request.getRequestDispatcher("UserGroup/groupManage.jsp").forward(
                        request, response);
            } else
            // 查看当前系统中所有的用户组信息
```

```
                if ("查看全部".equals(sub)) {
                        request.getSession().putValue("session", null);
                        response.sendRedirect("UserGroup/groupManage.jsp");
                } else
                // 向系统中添加新的用户组
                if ("添加".equals(sub)) {
                        boolean flage = gmi.insertGroup();
                        if (flage) {
                                request.setAttribute("flage", "用户组添加成功！");
                                request.getRequestDispatcher("UserGroup/addGroup.jsp").forward(
                                                request, response);
                        }

                        else {
                                request.setAttribute("flage", "用户组添加失败！可能是数据库中已存在相同
记录！");
                request.getRequestDispatcher("UserGroup/addGroup.jsp").forward(
                                                request, response);
                        }
                } else
                // 删除用户所选中的用户组里的信息
                if ("删除用户组".equals(sub)) {
                        boolean flage = gmi.delGroups();
                        if (flage)
                                response.sendRedirect("UserGroup/groupManage.jsp");
                        else {
                                out.print("多个用户组删除失败！");
                                out.print("<a onclick=\"javascript:history.go(-1)\">返回上一页</a>");
                        }
                } else
                // 修改用户组的基本信息
                if ("修改".equals(sub)) {
                        boolean flage = gmi.updateGroup();
                        if (flage)
        F．response.sendRedirect("UserGroup/groupManage.jsp");
                        else {
                                out.print("用户组修改失败！");
                                out.print("<a onclick=\"javascript:history.go(-1)\">返回上一页</a>");
                        }
                }
                // 删除单一的用户组
                else if ("1".equals(sign)) {
                        boolean flage = gmi.delGroup();
                        if (flage) {
```

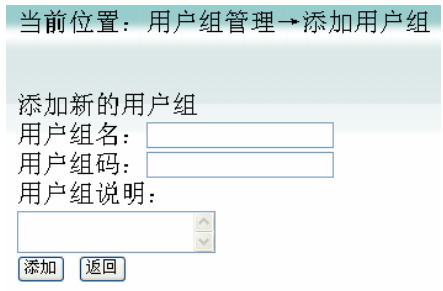
```
                    request.getSession().putValue("session", null);
                    response.sendRedirect("UserGroup/groupManage.jsp");
                } else
                    out.print("用户组删除失败！");
                out.print("<a onclick=\"javascript:history.go(-1)\">返回上一页</a>");
            }
            // 如果直接输入 URL 则提示先登录
            else {
                out.print(" 请先登录！");
            }
            out.flush();
            out.close();
        }

    G.  public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            doGet(request, response);
        }
    }
```

现在对以上代码进行分析。对于 JSP 的具体用法将会在第 7 章中讲到，在这里主要分析 Servlet 的用法。Servlet 的主要作用是获取 JSP 页面传过来的值，然后进行业务处理，处理完业务后进行页面跳转。Servlet 的本质是一个类，由于它继承了 HttpServlet 类而具备了获取页面表单数据及转发数据的能力。

在本案例中，Servlet GroupManageAction 通过关键字 extends 继承了 HttpServlet 类，如 Servlet 源代码中标记 A 处所示。在 GroupManageAction 中，包含了两种方法，一个是 public void doGet(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException { } 方法，如标记 B 处所示；一个是 public void doPost(HttpServletRequest request, HttpServletResponse response) { } 方法，如标记 G 处所示。在这两个方法中，都包含了两个参数。一个参数是 HttpServletRequest，该类对象主要是通过 HTTP 协议获取页面表单数据。如源代码中标记 C 处所示，通过 request 调用 getParameter() 方法而获取页面数据。在 String sign = request.getParameter("sign")中 getParameter 中的参数值 sign 为 JSP 页面中的参数名称或表单 name 属性值。通过此方式在 Servlet 中可以根据参数的名称或表单属性名称而获取所对应的值。该类对象也可以负责页面的跳转，如标记 E 处所示。另一个参数是 HttpServletResponse，该类对象主要是 Servlet 响应对象，内容可以是一段 HTML 代码，也可以通过一个方法实现页面响应。如源代码中标记 F 处，通过 sendRedirect("UserGroup/groupManage.jsp")方法而重定向到 UserGroup/groupManage.jsp 页面。

在源代码中 G 处为 doPost()方法。在该方法中，调用当前类 doGet(request, response)方法。这样做的主要目的是如果页面采用的提交方法为 post，也可以调用 doGet()方法，从而实现代码的业务逻辑，但最好的方式是在 doPost()方法中完成代码的业务逻辑。

在用户组管理首页及 groupManage.jsp 中，若单击“添加新用户组”按钮后，则会通过调用 JavaScript 语句 onclick="javascript:window.location='../../bank/UserGroup/addGroup.jsp'" 跳转到添加用户组页面 addGroup.jsp。其页面如图 5.14 所示。



5.14　添加用户组页面

添加用户组页面 addGroup.jsp 的主要源代码如下。

```
        <TITLE>addGroup.jsp</TITLE>
        </HEAD>
        <BODY>
        <%String flage=(String)request.getAttribute("flage");
        if(flage==null)
           flage=""; %>
        当前位置：<a onclick="javascript:window.location='../../bank/UserGroup/groupManage.jsp'">用户组管理</a>→添加用户组
        <!-- action 的值为控制该操作的 Servlet 的 URL，在 Servlet 中来控制页面跳转到指定的下一步页面-->
        <FORM action="../../bank/GroupManageAction" method="post">
        <font color="red"><%=flage %></font><br>添加新的用户组<BR>用户组名：<INPUT type="text" name="group_name" size="20"><BR>用户组码：<INPUT
              type="text" name="group_id" size="20"><BR>用户组说明：<BR>
        <TEXTAREA rows="2" cols="20" name="group_description"></TEXTAREA>
        <BR>
        <INPUT type="submit" name="sub" value="添加">
        <INPUT type="button" name="return" value="返回" onclick="javascript:window.location='../../bank
        /UserGroup/groupManage.jsp'"></FORM>
        </BODY>
        </HTML>
```

在该页面的表单中主要是要添加用户组名、用户组码、用户组说明等信息。由表单中的 action 属性可知，该页面所对应的 Servlet 为 GroupManageAction 。提交方法为 post 方法。如<FORM action="../../bank/GroupManageAction" method="post">。

由此可知，用户组管理页面和添加用户组页面所对应的 Servlet 都为 GroupManageAction。那么 Servlet 如何区分到底处理哪个请求页面呢？事实上在该 Servlet

中有条语句 String sub = request.getParameter("sub")，通过该条语句获取当前 Servlet 需要处理的页面参数，然后通过参数进行判断到底执行哪个请求页面数据。例如，在 addGroup.jsp 页面中，“添加”按钮的 name 属性值为 sub,它所对应的值为“添加”，这时一旦单击“添加”按钮如下代码将会将该 sub 所对应的值传入 Servlet GroupManageAction，在 GroupManageAction 中通过写如下代码便可处理添加用户组信息内容。

```
if ("添加".equals(sub)) {
        boolean flage = gmi.insertGroup();
        if (flage) {
            request.setAttribute("flage", "用户组添加成功！");
            request.getRequestDispatcher("UserGroup/addGroup.jsp").forward(
                    request, response);
        }
```

具体添加内容在 gmi.insertGroup()方法中完成。该方法返回一个布尔值，若成功则显示“用户组添加成功！”信息，否则跳转到添加用户组页面 addGroup.jsp 中重新添加新用户组信息。

根据前面例子可以知道，一个请求页面对应一个 Servlet,一个 Servlet 可以对应多个请求页面。在同一个 Servlet 中处理多个页面请求时需要通过参数名加以区分。

5.7　实训操作

在此次实训中，主要是要熟练掌握 Servlet 的执行原理，HTML 页面的数据或 JSP 页面的数据被如何传给服务器端，服务器端怎样接收数据，数据处理完后又怎样进行页面的跳转。现在用一个例子完成从 HTML 页面传递数据到服务器端 Servlet 及服务器端处理完业务后跳转页面的功能。

现在以人力资源管理系统的用户管理模块为例。图 5.15 所示为用户管理页面，主要功能包括按条件查询用户信息、删除被选中的用户信息、编辑用户信息。图 5.16 所示为在用户管理页面中单击“添加用户信息”按钮后弹出的新增用户信息页面。通过该模块超级管理员可以设置一般管理员的权限，管理该系统的后台信息。该模块的功能包括：①添加新用户；②为用户设置姓名、性别、登录账号、登录密码、确认密码；③删除用户；④密码修改，单击“编辑”按钮可以进行密码修改；⑤分配角色，0 为超级管理员，1 为普通管理员。

图 5.15　用户管理页面

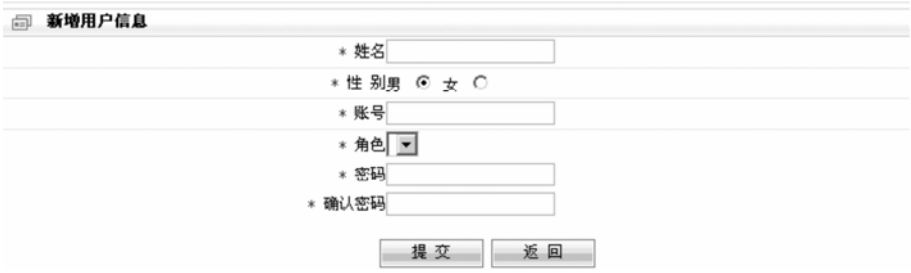


图 5.16　新增用户信息页面

现在写一个 Servlet 专门处理用户管理及新增用户信息。在 Servlet 中需要接收查询条件姓名和账号。Servlet 接收数据的语句为 request.getParameter(" ")。然后，单击“查询”按钮在页面中显示满足条件的所有数据。在此 Servlet 中可以写 4 个方法分别完成查询、清空数据、删除、编辑等功能。

习　题

一、单项选择题

1．以下有关 Servlet 处理流程步骤的表述不正确的是（　　）。

A．客户端发送一个请求给服务器端。服务器将请求信息发给 Servlet。

B．Servlet 引擎，也就是 ejb 容器会调用 Servlet 的 service()方法。

C．Servlet 构建一个响应，并将其传递给服务器。这个响应是动态建立的，相应的内容通常取决于客户端的请求，在这个过程中也可以使用外部资源。

D．服务器将响应返回给客户端。

2．以下是 web.xml 文档的一部分。

```
<servlet>
        <servlet-name>Display</servlet-name>
        <servlet-class> DisplayServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet>
        <servlet-name>Search</servlet-name>
        <jsp-file>/search/search.jsp</jsp-file>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

请问以上 web.xml 文档中的设置是指示服务器首先装载和初始化哪个 Servlet?（　　）

A．Display

B．DisplayServlet

C．search.jsp

D．由 search.jsp 生成的 Servlet

3．以下有关 Servlet 常用请求头含义的说法错误的是（　　）。

A．Accept-Language 头指出客户端倾向的语言，如 en。

B．Content-Length 头以字节为单位给出 post 数据的长度。

C．Host 表明了主机和端口。

D．User-Agent 头表明了发出请求的客户代理的信息。

4．以下关于 Java Servlet API 说法错误的是（　　）。

A．Java Servlet API 是一组 Java 类，它定义了 Web 客户端和 Web Servlet 之间的标准接口。

B．Java Servlet API 由两个包组成：javax.servlet 和 javax.servlet.http。

C．javax.servlet.http 包对 HTTP 协议提供了特别的支持。

D．javax.servlet 包提供了对除 HTTP 协议外其他协议的支持。

5．下列有关 ServletRequest 和 ServletResponse 的描述错误的是（　　）。

A．Servlet 引擎使用 ServletRequest 来向 Servlet 提供有关客户端请求的信息，使用 ServletResponse 向客户端传送经过 MIME 编码的数据。

B．HttpServletRequest 类和 HttpServletResponse 类能够提供与协议相关的数据。Servlet 的 Service()方法的参数是 servletRequest 对象或 ServletResponse 对象中的一个。

C．发送文本数据时，使用 getWriter()方法返回 PrintWriter 对象；发送二进制数据时，使用 getOutputstream()方法返回 ServletOutputStream 对象。

D．在调用 getWriter()或 getOutputStream()方法之前必须调用 setContentType()方法。

6．在 Servlet 中输出采用下面哪一个对象？（　　）

A．request

B．response

C．out

D．page

二、多项选择题

1．以下有关 Cookie 的说法正确的是（　　）。

A．Cookie 是将会话数据保存在客户端来维护会话状态的一种方式。它是服务器发送给浏览器的体积很小的纯文本信息。

B．获得 Cookie 后，用户以后再访问同一个 Web 服务器时，浏览器会把 Cookie 原样发送给服务器。

C．Cookie 是服务器端状态管理机制。

D．Cookie 是相对安全的一种状态管理机制。

2．以下哪些方法是 Servlet 的生命周期接口定义的？（　　）

A．init()
B．service()
C．destroy()
D．create()

3．下面哪些是页面跳转的常用方法？（　　）
A．forward()
B．service()
C．include()
D．create()

4．下面哪种说法是正确的？（　　）
A．get 方法对上传数据量有限制。
B．post 方法对上传数据量无限制，可为大文件上传。
C．get 方法既可以采用 URL 地址栏方式，也可以采用表单方式。
D．post 方法只能采用表单方式。

三、编程题

1．创建 Servlet 用来显示你的基本信息，包括班级、学号、姓名；要求姓名和班级用中文显示，能正确地显示中文信息，编译部署；修改配置文件，使得用 http://127.0.0.1:8080/Myservlet 可以访问到你创建的 Servlet。

2．使用 Servlet 输出个人信息，包括姓名、班级、学号、住址、年龄、Email。

3．建立 login.html，用户可以输入姓名、学号、Email 信息，通过表单提交信息，并使用 Servlet 进行用户合法性控制,如果姓名和学号正确显示成功信息，显示“欢迎你”；如果姓名和学号错误则跳转到 invalid.jsp 页面，提示该姓名和学号不匹配。（可以连接数据库来判断。）

4．使用 httpSession 显示该会话过程中所有用户的登录次数。

5．累积用户数据，类似购物车，使用 ArrayList 作为属性类型，在添加页面添加用户数据，在显示页面显示用户的所有数据。

第 6 章　JavaScript技术

📖 知识点

- JavaScript 的有关概念和特点
- JavaScript 的基本语法
- JavaScript 的事件处理机制及如何与 HTML 进行交互
- JavaScript 的对象
- JavaScript 的编程技巧

📣 难点

- JavaScript 的编程技巧
- JavaScript 的事件处理机制

✍ 掌握

- JavaScript 的用法
- JavaScript 的编程技巧

任务引入

在 Java Web 项目中，经常会遇见显示层表单数据的输入验证和下拉菜单、下拉列表的生成等功能。而这些功能可以将数据传输给服务器端，服务器再将数据传给 Servlet 处理，Servlet 处理完数据后再将数据传输给客户端。若只是表单数据的输入验证或菜单列表的生成就需要经过这么多步骤，那计算机的执行速度也就太慢了。为了解决上述问题，Internet 采用 JavaScript 脚本语言，使得表单数据不需要被传递给服务器端而直接在客户端进行验证，一些下拉菜单或下拉列表也可以直接在客户端完成而不需要服务器的处理，从而提高了数据的访问速度。

本章主要讲解 JavaScript 的概述、JavaScript 的基本语法、JavaScript 的事件处理机制及 JavaScript 的对象等重要内容。

6.1　JavaScript概述

随着因特网的迅速发展与盛行，网站可以说是多得数不清。网站的功能也不再局限于浏览，还需要和用户进行交互。JavaScript 的出现就满足了这种需求。JavaScript 是目前在因特网上应用最为广泛的客户端的脚本编程语言。它使得信息和用户之间不只是一种显示和浏览的关系，而是实现了一种实时的、动态的、可交互的功能。

使用 JavaScript 脚本是很安全的，因为 JavaScript 脚本不允许访问本地硬盘，而且不能将数据存入服务器中，不允许对网络文档进行修改。

JavaScript 最开始主要是用于在提交给服务器端的程序一些表单数据之前执行简单的核查，即表单确认。现在其用途慢慢发展到给网页页面作修饰。JavaScript 不仅可以完成简单的小动画，还能构造复杂的导航系统，如下拉菜单或下拉列表。

用 JavaScript 编写的脚本程序是作为 HTML 网页的一部分存在的，由浏览器直接解释

执行，所以 JavaScript 具有平台无关性。只要客户端的浏览器支持 JavaScript，网页就可以正常运行。现在的主流浏览器 Internet Explorer 就支持 JavaScript。

6.1.1　如何将JavaScript脚本嵌入到HTML文档中

编写 JavaScript 脚本和写网页很类似，可以用任何一种文字编辑器来编写，与 HTML 文档配合，将其插入到HTML 文档中，然后执行HTML 文档即可。在HTML 中添加JavaScript 脚本可以使用 3 种方法：包含在<script>…</script>内；通过<script>标记的 src 属性成为链接文件；包含在 HTML 事件处理属性中，如 onclick。

1．在<script>…</script>内加入JavaScript脚本

在 HTML 文档中嵌入 JavaScript 脚本的基本方法是使用<script>标记。通常指明使用的脚本语言的方法是在 language 属性中进行具体的设置，例如：

```
<script language="JavaScript">
</script>
```

然而，根据 W3C HTML 语法，language 属性不应该使用。作为替换，应该设置 type 属性来表明要使用的语言的 MIME（Multipurpose Internet Mail Extension，多用途因特网邮件扩展，也就是多媒体文件格式）。JavaScript 的 MIME 类型是 text/javascript。其一般格式如下。

```
<html>
    <script type= "text/javascript">
        添加 JavaScript 代码
    </script>
</html>
```

例 6_1：利用 JavaScript 显示信息“Hello Everyone！欢迎进入 JavaScript 世界！”。

```
<html>
<body>
    <script type="text/javascript">
        document.write("Hello Everyone！欢迎进入 JavaScript 世界！");
    </script>
</body>
</html>
```

以 html 为文件扩展名保存此文件，在浏览器中打开此文件，查看实际效果。运行结果如图 6.1 所示。

图 6.1　在<script>…</script>内加入 JavaScript 脚本的效果 JavaScript

说明：当浏览器打开 HTML 文档时，会顺序解释整个文档，普通的脚本语句在被浏览器扫描到时执行。在例 6_1 中，document 是文档对象，它代表当前在浏览器窗口中打开的文档，使用文档对象可以访问页面上的各种元素。document.write()是文档对象的 write 方法()，表示向文档中写入内容。

2．利用<script>标记的src属性

利用<script>标记的 src 属性，可以在网页中使用外部的 JavaScript 脚本文件。这样可以提高代码的使用效率。当需要在不同的网页中使用相同的脚本代码时，可以将这些需要共享的代码写成一个文件，以 js 为扩展名进行保存。然后用<script>标记的 src 属性来引用。例如：

```
<script type= “text/javascript” src= “Ex6_2.js”>
</script>
```

其中，src 的值是 Ex6_2.js。这个值是一个指向外部脚本的 URL 路径。在这个例子中，它们在同一个目录中，但是也可以通过绝对路径来实现链接。

例 6_2：利用 JavaScript 的<script>标记的 src 属性,显示信息“Hello Everyone！欢迎进入 JavaScript 世界！”。

首先在记事本中新建一个文件，以 Ex6_2.js 存盘。在文件中写入：

```
window.alert("Hello Everyone！欢迎进入 JavaScript 世界！");
```

然后建立 HTML 文档，代码如下。

```
<html>
<body>
  <script type="text/javascript" src="Ex6_2.js">
  </script>
</body>
</html>
```

以 Ex6_2.html 为文件名保存此文件，在浏览器中打开此文件，查看实际效果。运行结果如图 6.2 所示。

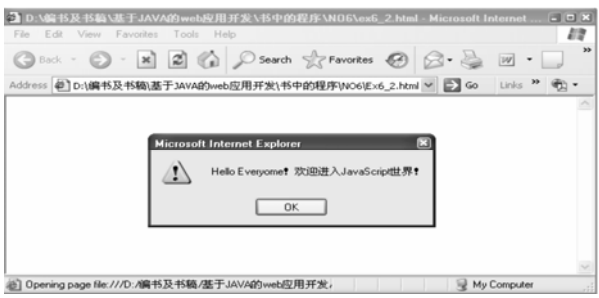


图 6.2　利用 JavaScript 的<script>标记的 src 属性加入 JavaScript 脚本的效果

说明：在 js 文件中，只能包含 JavaScript 语句，不能包含其他语句和标记，也不能将<script>...</script>写入该文件中，否则会出现错误。window.alert(text)是窗口对象的 alert()的方法，指的是弹出提示对话框。text 参数为对话框中的提示信息。

3．利用HTML的事件处理属性

为了使一个页面更具有交互性，用户可以添加 JavaScript 命令，它可以等待用户的特定行为。为了指明这些脚本，用户可以设定不同的事件处理属性。通常通过设定 HTML 的属性来引用一个脚本，可以将这些 HTML 属性统称为事件处理属性。所有这些属性都以单词“on”为开头。

例 6_3：利用事件处理属性显示信息“Hello Everyone！欢迎进入 JavaScript 世界！”。

```
<html>
<head>
<script type="text/javascript">
    function showMessage()
    {
        alert("Hello Everyone! 欢迎进入 JavaScript 世界！");
    }
</script>
</head>
<body>
<form>
  <input type="button" name="textBtn" value="请按钮" onclick="showMessage()">
</form>
</body>
</html>
```

以 Ex6_3.html 为文件名保存此文件，在浏览器中打开此文件，查看实际效果。运行结果如图 6.3 所示。

图 6.3 利用事件处理属性加入 JavaScript 脚本的效果

说明：showMessage()为 JavaScript 中自定义的函数。其调用是在单击按钮“请按钮”时执行的。通过设置“请按钮”按钮的属性 onclick="showMessage()"来定义这种关联。onclick 是鼠标单击事件。

与普通脚本语句的顺序执行不同，函数只有在被调用时才执行，一种是写代码调用，另一种是隐含调用，通过事件处理机制。例如，上例中 showMessage()函数的调用只有在用户单击“请按钮”时才会执行，它不会在浏览器打开该 HTML 后自动执行。正是这种自动调用的事件处理机制，使得 HTML 文档具有交互性。

6.1.2　JavaScript脚本在HTML中的位置

由前面可以知道，在 HTML 中加入 JavaScript 脚本可以使用 3 种方法。那么，JavaScript 脚本究竟被添加到 HTML 中的什么地方？

JavaScript 脚本可以被添加到 HTML 中<html>…</html>内的任何地方。并且可以在 HTML 中添加多个 JavaScript 脚本代码。

但是需要注意的是，如果在 JavaScript 中定义了函数，那么就必须遵循先定义后使用的原则，否则浏览器无法执行该函数，将会弹出出错对话框。为了避免发生这种错误，一般把函数定义放在<body>…</body>前、<head>...</head>中，这样确保在调用函数时，已经先定义函数了。

6.1.3　事件处理机制

所谓事件，就是用户执行的某种操作。例如，鼠标单击、双击都是事件。事件响应就是对这种操作所做出的响应。例如，在例 6_3 中单击按钮打开对话框就是对用户单击“请按钮”按钮进行的响应。

在浏览器中，所有事件都由浏览器进行检测，当浏览器检测到某个事件要发生时，就会自动寻找相应的事件响应函数来响应这个事件。事件响应函数就是在事件发生时自动执行的函数。一般，事件响应函数的名称是在事件名前加 on。例如，鼠标单击事件的响应函数是 onclick()。浏览器正是通过对事件的响应来实现与网友浏览者的交互的，而 JavaScript 脚本的应用就在于编写事件响应函数，来动态响应用户的操作，动态改变页面的内容。

6.2　JavaScript的基本语法

每种语言都有其语法结构，JavaScript 也不例外。由于 JavaScript 是从 C 语言演变而来的，所以 JavaScript 和 C 言语有许多相似的地方。例如，语句都是以分号作为结束，注释都是有两种“//”和“/*……*/　”。这给学习带来很多方便。

6.2.1　数据类型及变量

如果学过其他编程语言的话，就会知道有多种数据类型，不同的数据类型有不同的形式，也有不同的使用方法。在此介绍较为常用的几种数据类型。

1．数据类型

（1）数值型

数值型包括整型和浮点型。整型有 3 种表示方法：十进制表示法，如 12，23，54；八进制表示法，以英文字母 O 开头，0～7 的数字来表示，如 O12，O34，O54；十六进制表示法，以 0X 开头，0～9 的数字和 A～F 的英文字母来表示，如 0X76，0X6A，0XBF。

浮点数就是小数。在 JavaScript 中，浮点型有两种表示方法：第一种是通常的小数表示法，如 88.5，98.4；另一种是指数表示法，如 5.43E2，−12E−3。

（2）字符串类型

在 JavaScript 中，字符串是用单引号或双引号括起来的多个字符，如“Welcome ！”、“Hello！张三”。注意：在 JavaScript 中没有字符类型。

（3）布尔类型

布尔类型的值只有两个，即表示逻辑真的 true 和表示逻辑假的 false。

2．变量

JavaScript 的变量名必须由数字、字母、下画线组成，并且不能以数字开头。例如，First1_one 和 e3 都是合法的变量名，而 4dd 和 first one 就不是合法的变量名。变量名是区分大小写的。例如，变量 A 和 a 是两个不同的变量。

关键字是 JavaScript 内部定义的具有特殊含义的字符，如 int，double 等。关键字不能被用做变量名。

声明一个变量用 var 关键字，其一般格式如下。

```
var 变量名;
```

定义变量的同时可以对变量进行赋值，这叫做变量的初始化。其格式如下。

```
var 变量名=值;
```

此时，变量的类型由变量的取值来决定。例如：

```
var a=23.5;//a 为浮点型
var s= "hello";//s 为字符串型
```

注意：

- 变量可以不用 var 声明，就可以使用。
- 可以先赋一种类型的值给变量，然后根据需要赋其他类型的值。例如：

```
s=56;
s="dsfsdf";
```

3．类型转换

在 JavaScript 中，如果表达式中运算符两边变量的类型不一致，就会进行自动的类型转换，将右边变量的类型转换成左边变量的类型。例如：

```
a=12;
s= "34";
var f=s+a;
var m=a+s;
```

给 a 赋值 12，给 s 赋值“34”，s+a，把 a 的类型转换成 s 的类型，即将整型 12 转换成字符串“12”，f 的值为字符串“3412”，而 a+s 则要把 s 的类型转换成 a 的类型，即将字符串“34”转换成整型 34，m 的值为整型 46。

如果 s 的值不是由数字形成的字符串，而是由字母等其他字符形成的字符串，如 var n=34+"sdf"，则会出现无法将字符串转换成整型的情况，这时就需要进行强制类型转换。强制类型转换是采用 JavaScript 提供的函数来实现的。

- parseInt()在字符串中提取一个整数，当遇到非数值型字符时停止。例如，parseInt("546sdf")的返回值为 546。
- parseFloat()在字符串中提取一个浮点数，即包括小数点，当遇到非数值型字符时停止。例如，parseFloat("56.65dfd")的返回值为 56.65。

例 6_4：数据类型转换练习。

Ex6_4.js

```
var a="54";
b=78;
s="sdf";
d="56dfgd";
var m=s+b;
var n=b+parseInt(d);
var e=a+b;
document.write("m 为字符串加整型,结果为字符串 m="+m);
document.write("n 为整型加字符串，需要进行强制类型转换,结果为整型 n="+n);
document.write("e 为字符串加整型，结果为字符串 e="+e);
```

Ex6_4.html

```
<html>
<body>
  <script type="text/javascript" src="Ex6_4.js">
  </script>
</body>
</html>
```

程序的运行结果如图 6.4 所示。

图 6.4　数据类型转换

4．数组

数组是指将相同类型的数有序地排在一起。在 JavaScript 中，数组声明是用 new 关键字及 Array()方法所组成的。new 是指创建一个新的对象，Array()是声明一个数组。例如：

```
a=new Array("month", "march", "may", "December");
```

定义数组 a ，含有 4 个元素，类型为字符串。这 4 个元素分别是：a[0]=“month”，a[1]=“march”，a[2]=“may”，a[3]=“December”。

注意：数组的下标是从 1 开始的。

6.2.2　运算符与表达式

JavaScript 中的运算符很多，最常用的包括算术运算符、关系运算符和逻辑运算符。其运算与 C 语言相似。各种操作符分别如表 6.1、表 6.2、表 6.3 所示。关系运算符的结果是一个布尔值，true 或 false。

表 6.1 算术运算符

算术运算符	描　　述
＋	加
－	减
*	乘
/	除
%	取余，可以用于浮点数
++	自加
--	自减

表 6.2　关系运算符

关系运算符	描　　述
＞	大于
＜	小于
＞＝	大于等于
＜＝	小于等于
＝＝	等于
！＝	不等于

表 6.3 逻辑运算符

逻辑运算符	描　　述
&&	逻辑与，当两个操作数均为 true 时，结果才为 true
‖	逻辑或，当其中任一操作数为 true，结果就为 true
！	逻辑非，对操作数取反，true 变 false，false 变 true

运算符之间的优先级为：!、算术运算符、关系运算符、&&、||、赋值运算符。

条件运算符是唯一的一个三目运算符，其形式如下。

```
条件? 表达式 1：表达式 2
```

若条件的值为真，结果为表达式 1 的值；若条件的值为假，结果为表达式 2 的值。

JavaScript 中的表达是用运算符和括号将运算对象（也称为操作数）连接起来、符合 JavaScript 语法规则的式子。

6.2.3 程序结构

JavaScript 的程序结构包括顺序结构、选择结构和循环结构。

1．顺序结构

在顺序结构中，语句会按顺序一条一条地被执行。

2．选择结构

在选择结构中，语句根据条件判断可能被执行一次，也可能不被执行。用条件语句可以构成选择结构。

条件语句包括 if 语句和 switch 语句。

（1）if 语句

if 语句的格式有以下 3 种。

```
if (条件)  { if 子句; }
```

满足条件，执行 if 子句，否则程序向下执行。

```
if(条件)
      { if 子句; }
else
       {else 子句; }
```

为二选一结构。条件为 true，执行 if 子句；条件为 false，执行 else 子句；然后程序向下执行。

```
if(条件 1)
       { if 子句 1;}
    else if(条件 2)
       {if 子句 2;}
    else if(条件 3)
       {if 子句 3;}
          …
          …
          …
    else if(条件 n)
       {if 子句 n;}
    else
       { else 子句;}
```

此结构为多选一结构，首先判断条件 1，若为 true 则执行 if 子句 1，然后跳出 if 语句，程序向下进行；否则对条件 2 进行判断，若为 true，则执行 if 子句 2，然后跳出 if 语句；若为 false，继续向下进行判断。如果有一个条件成立，则执行该 if 子句；如果条件均不成立，则执行 else 子句，然后程序向下执行。

（2）switch 语句

switch 语句也为多选一结构，它要比 if 语句的多选一结构更清楚明了。其一般格式如下。

```
switch(表达式)    //表达式的值为数值型或字符串型
{
    case 值 1：语句 1；[ break；]    //值 1 为表达式的可能取值
    case 值 2：语句 2；[ break；]    //值 2 为表达式的可能取值
     …
case 值 n：语句 n；[ break；]        // 值 n 为表达式的可能取值
default：语句 n+1；
}
```

首先计算表达式的值，在 case 后的值中查找其结果。若找到，则执行该 case 后的语句，然后 break 跳出；若没有 break，程序会依次执行余下的 case 后的语句，直到碰到 break 跳出。若没有找到，则执行 default 后语句 n+1。

例 6_5：if 语句的应用。根据成绩打印等级，90 分以上为 A 等级，80～89 分为 B 等级，70～79 分为 C 等级，60～69 分为 D 等级，60 分以下为 F 等级。程序如下。

```
<html>
<head>
<script type="text/javascript">
 function fun()
 {
    var g=parseInt(frm.num.value);
    if(g>=90)
       alert("您的成绩达到 A 等级");
    else if(g>=80)
       alert("您的成绩达到 B 等级");
    else if(g>=70)
       alert("您的成绩达到 C 等级");
    else if(g>=60)
       alert("您的成绩达到 D 等级");
    else
       alert("您的成绩不及格为 F 等级");
 }
</script>
</head>
<body>
<center>
```

```
<font color="red" size="4">
    成绩报表
</center>
<form name="frm">
  请输入您的成绩：
  <input type="text" name="num" size="10"><br><br>
  <input type="button" value="确定" name="btn" onclick="fun()"><br><br>
</form>
</body>
</html>
```

以 Ex6_5.html 为文件名保存程序，程序的运行结果如图 6.5 所示。

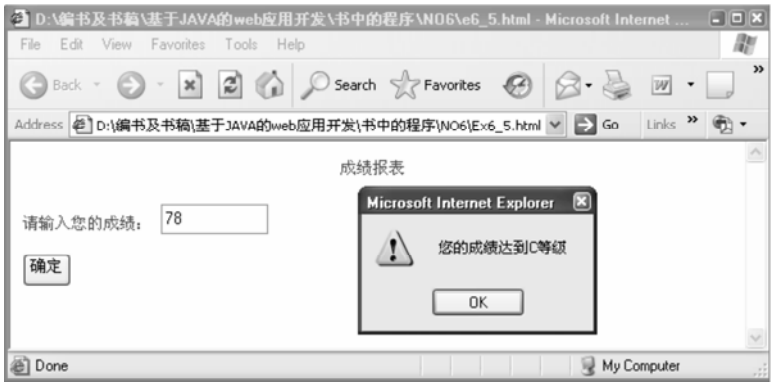


图 6.5　if 语句的应用

3．循环结构

在循环结构中，有些语句需要被执行多次。用循环语句可以构成循环结构。

JavaScript 中的循环语句包括 while 语句、do_while 语句和 for 语句、for_in 语句。要写一个正确的循环语句，需要对循环控制变量做了件事。第一，给循环控制变量赋初值，它通常被放在循环的外面；第二，写出关于循环控制变量的结束条件，它的位置根据循环语句的不同位置相对固定；第三，循环控制变量的递增/递减变化，它通常被放在循环的内部。下面对循环语句进行详细描述。

（1）while 循环语句

while 循环语句的一般格式如下。

```
赋初值
while (循环条件)
{
    循环语句;
    递增/递减;
}
```

while 循环又被称为当型循环。当条件成立时，执行循环体；当条件不成立时，退出循

环。它是先判断后执行的循环，所以有可能循环一次也不执行。

（2）do_while 循环语句

do_while 循环语句的一般格式如下。

```
赋初值
do
{
    循环语句;
    递增/递减;
}while(循环条件)
```

do_while 循环又被称为直到型循环。首先执行循环体一次，然后进行判断，如果条件成立，继续循环，直到循环结束。它是先执行后判断的循环，循环至少执行一次。

（3）for 循环语句

for 循环语句的一般格式如下。

```
for(初值；循环条件；递增/递减)
{
    循环语句;
}
```

for 循环通常被用于循环次数事先确定的情况，而 while 和 do_while 循环通常被用于不知道循环次数的情况。例如，编写程序，从键盘输入若干字符，以 “?”结束，统计数字字符的个数。对这个程序，就需要用 while 或 do_while 循环。

（4）for_in 循环语句

for_in 循环语句的一般格式如下。

```
for(变量 in 对象或数组)
{
   循环语句;
}
```

此循环的范围是一个对象的所有属性或一个数组中的所有元素。

例 6_6：编写程序，输入任意一个正整数 N，计算 N 的所有因数之和。

```
<html>
<head>
<script type="text/javaScript">
    function fun()
    {
        var sum=0,i=0;
        number=frm.num.value;
        while(i<=number)
        {
```

```
            i++;
            if(number%i!=0) continue;
            sum=sum+i;
        }
        alert(sum);
    }
</script>
</head>
<body>
  <form name="frm">
      N：<input type="text" name="num" size="10"><br><br>
          <input type="button" name="btn" value="确定" onclick="fun()"><br><br><br>
      <font color="blue" >
          输入任意一个正整数 N，计算 N 的所有因数之和
  </form>
</body>
</html>
```

以 Ex6_6.html 为文件名保存程序，程序运行结果如图 6.6 所示。

图 6.6　循环的应用

6.3　JavaScript的对象

在 JavaScript 中有两类可以直接使用的对象，即内置对象和浏览器对象。内置对象包括同基本数据类型有关的对象，如数组（Array）对象、字符串（String）对象，日期（Date）对象等；它允许用户自己创建对象。浏览器对象是浏览器在装载网页时创建的对象，通过这些对象可以控制页面元素的显示，如表单、窗口等。例如，prompt()弹出一个 JavaScript 提示对话框。浏览器对象中最高层的对象是窗口（window）对象，它代表当前的浏览器窗口。在窗口对象下有文档（document）、位置（location）、历史（history）等对象；在文档对象下有表单(form)、图像(image)和链接(link)等多种对象；在表单对象下有按钮(button)、复选框（checkbox）等对象。

6.3.1　Array，Date，String对象

创建对象采用 new 操作符。其一般格式如下。

```
var 对象实例名=new 对象名(参数列表)；//参数若为多个，参数之间用分号隔开；若没有参数，()也必不可少。
```

引用对象的属性和方法需要采用如下的格式。

```
对象实例名．属性名；
对象实例名．方法名(参数列表)；//参数若为多个，参数之间用分号隔开；若没有参数，()也必不可少。
```

Array 对象可以直接增加和修改数组元素，使用 delete 命令删除元素。Array 对象的各种方法如表 6.4 所示。

JavaScript 提供了一系列的读写方法对日期的每一个字段进行读写，并且可以将日期转化为字符串和将字符串转化为日期。Date 对象的常用方法如表 6.5 所示。

在建立网页时,经常会对用户输入的信息进行检查,看看用户使用的格式是否正确，例如，用户输入的年龄只能是数字字符,不能是字母等其他字符。这就用到字符串处理函数。处理字符串的对象是 String 对象，所有与字符串相关的方法、属性都在此对象中。String 对象的常用方法可以分为两大类：一类是关于字符串的处理，另一类是关于字符串的显示。String 对象的常用方法如表 6.6 所示。

表 6.4　数组（Array）对象的常用方法

方　法	说　明
length	数组的长度
concat ()	两个数组进行连接。例如，a=new Array("abc"); b=new Array("xyz");c=a.concat(b); 则 c=("abc"，"xyz")
toString()	把数组中所有元素连接成一个字符串
join ([sp])	把数组中所有元素连接成一个字符串，sp 为连接时使用的分隔符。默认为逗号
pop ()	返回数组中的最后一个元素，并把此元素删除，数组长度减 1
push (参数)	给数组中添加元素到最后
shift ()	删除并返回数组的第一个元素，并且后面的元素依次向前移一位
unshift(参数)	可将多个元素添加到数组的开头，原有元素依次向后移
reverse ()	将整个数组中的元素顺序倒转
splice ()	任意加入或取代数组中的元素

表 6.5　日期（Data）对象的常用方法

构造方法

Date(年，月，日)	仅输入日期，时间设为 0 时 0 分 0 秒。例如，thisDay=new Date(2008,7,20)
Date(年,月,日,时,分,秒)	例如，thisDay=new Date(2008,7,20,10,23,11)

续表

Date(月日,年 时:分:秒)	例如，thisDay=new Date(May 23,2006 12:23:45)
提取年、月、日、时、分、秒函数(没有参数)	
getYear(),getMonth(),getDate()	从日期变量中取出年、月、日
getDay()	返回是本周的第几天（返回 0～6），0 代表星期日
getHours(),getMinutes(),getSeconds()	返回日期的时、分钟、秒
getTime()	以毫秒为单位，返回自 1970 年 1 月 1 号 0 时 0 分 0 秒到该日期所指定时间所经过的毫秒数
设置年、月、日、时、分、秒函数	
setYear(年)	
setMonth(月)	输入值 0～11 分别代表 1 月到 12 月
setDate(日期)	输入值 0～31
setHours(小时)	输入值 0～23

表 6.6　字符串（String）对象的常用方法

字符串处理函数	
方　　法	说　　明
length	求字符串的长度，中英文字符长度都是 1。例如，s="asd 大小"，则 s.length=5
charAt(位置)	找出指定位置的字符，位置从 0 开始。例如，s="asd 大"，则 s.charAt(2)= "d"
indexOf(串，位置)	从指定位置开始查找指定字符，找到，返回位置；没有找到，返回−1。例如，s="abc" ，则 s.indexOf("b")=1，s.indexOf("a",1)=−1
lastIndexOf(串，位置)	方法同 indexOf，区别是从后向前查找
substring(起始，结束)	取出从起始位置到结束位置的字符串。例如，s="abdfd"，则 s.substring(1,3)= "bdf"
toLowerCase()	大写变小写。例如， s="JavaScript*"，则 s.toLowerCase()="javascript*"
toUpperCase()	小写变大写
字符串显示函数	
fontsize(字号)	改变字符串的字体大小。如 s="字体字号 2"，则 s.fontsize(2)
fontcolor("字体颜色")	如 s="红色字体"，则 s.fontcolor("red")
big()　small()	字体变大、变小
bold()　italics()	字体变成粗体、斜体
blink()	字符串输出时闪烁
strike()	字符串输出时在文字中间加上一条线
sub()　sup()	字符串输出时变成下标（上标）文字

6.3.2　窗口对象

在网页上用 JavaScript 设计程序，一个目的就是和用户进行互动。要想互动，就必须知道用户在网页上输入了哪些信息。读取信息后进行处理，然后把处理的结果显示于浏览器上。在 JavaScript 中要读取用户输入的信息，就必须通过窗口对象来读取。在窗口对象中，分为属性（Property）、方法（Method）和事件（Event）。在本节中，将介绍信息的读取方法，以及其他关于窗口的一些信息。

1．读取信息

在读取一个窗口中的某一个对象的某个属性时，一般采用如下格式。

```
document．窗口名称．对象名称．属性;
```

或

```
window．document．窗口名称．对象名称．属性;
```

window 是整个浏览器窗口的统称，它是最上层的唯一对象，所有浏览器上的对象都是 window 对象的一部分，所以写程序时 window 可以省略。document 是 window 下的一个子对象，这个对象包含所有在浏览器上看到的信息。窗口名称是写在<form name=“”>中的 name 值。对象名称则是其中某一个对象的名称，如按钮、文本框等。属性就是该对象的相关属性。举例说明，定义一个窗口如下。

```
<form name= "frm">
  <input type= "text" name= "nameTxt">
  <input type= "text" name= "passwordTxt">
  <input type= "button" name= "okBtn">
</form>
```

采用如下语句读取其中对象的属性值。

```
document.frm.nameTxt.value;
document.frm.passwordTxt.value;
document.frm.okBtn.value ;
```

2．open()方法和close()方法

open()方法的格式如下。

```
window.open(地址，窗口名称,[窗口特征]);
```

该方法的功能是在一个浏览器窗口中打开一个新的窗口来显示新的页面。地址是要链接的新网页的路径，可以是一个以 http 开头的完整主机及文件名称，也可以只是一个文件名称。窗口名称则是给这个新窗口的一个名称。窗口特征为新窗口的各属性的选项列表。选项 directories（目录按钮）、toolbar（工具栏）、location（地址栏）、status（状态栏）、menuBar（菜单）、scrollbars（滚动条）、resizeable（窗口缩放）取值 yes（1）或 no（0）。选项 width 和 height 为整数值，代表宽和高。

用 open()方法打开的窗口，就可以用 close()方法关闭。close()方法的格式如下。

```
window.close()
```

例 6_7：open()的应用。

```
<html>
<head>
<script src="text/javascript">
    function newOpen()
    {
        tb="toolbar="+frm.toolbar.value;
        l="location="+frm.location.value;
        s="status="+frm.status.value;
        m="menubar="+frm.menuBar.value;
        r="risizeable="+frm.resizeable.value;
        h="height="+frm.height.value;
        w="width="+frm.width.value;
        sum=tb+","+l+","+s+","+m+","+r+","+h+","+w;
        winObject=window.open("http://"+frm.InternetTxt.value,frm.winName.value,sum);
    }
</script>

</head>
<body>
<center>
  <font color="blue" size="20">打开新窗口</font>
  <form name="frm" action="">
      网址:http://<input type="text" name="InternetTxt" size=50 value=""><br>
      窗口名称：<input type="text" name="winName" size=50 value=""><br>
      工具栏：<select name="toolbar">
                  <option value="yes">是
                  <option value="no">否
              </select><br>
      地址栏：<select name="location">
                  <option value="yes">是
                  <option value="no">否
              </select><br>
      状态栏：<select name="status">
                  <option value="yes">是
                  <option value="no">否
              </select><br>
      菜单：  <select name="menuBar">
```

```
                    <option value="yes">是
                    <option value="no">否
                </select><br>
            窗口缩放：<select name="resizeable">
                    <option value="yes">是
                    <option value="no">否
                </select><br>
            窗口宽度：<input type="text" name="width" size=10 value=500><br>
            窗口高度：<input type="text" name="height" size=10 value=500>
            <br><input type="button" onclick="newOpen()" value="打开">
        </form>
    </center>
    </body>
    </html>
```

3．弹出对话框方法

弹出对话框方法共有 3 种：弹出警告对话框方法 alert()、弹出确认对话框方法 confirm()弹出提示输入对话框方法 prompt()。

alert()方法的功能就是显示一些信息给用户，其格式如下。

```
alert(字符串);
```

当 alert()方法被执行时，字符串中的文字便会出现在一个小的对话框中。

使用 alert()方法只显示一些信息给用户，而使用 confirm()方法则会弹出询问用户的一个对话框。输入的选择有两个，即确定和取消。由于这个方法是让用户确认用的，所以必须有返回值，若按下“确定”按钮，返回值为 true；若按下“取消”按钮，返回值为 false。其一般格式如下。

```
变量=confirm(提示字符串);
```

注意：变量值只有两个，true 或 false。提示字符串是对用户所发出的询问。

使用 prompt()方法可以让用户在对话框中输入一些文字，由于在这个方法中有用户输入的信息，所以就必须有一个变量来存储用户输入的数据。除了可以输入信息外，在提示输入对话框中也有与确认对话框中一样的两个按钮，分别是“确定”和“取消”。当用户在对话框中输入完数据之后，若按下“确定”按钮，浏览器就会把输入的数据返回；若按下“取消”按钮，则会返回一个空的字符串。这个方法有两个参数，一个是提示用户输入的字符串，一个则是默认值。其一般格式如下。

```
变量=prompt("提示字符串", "默认值");
```

变量中所存储的数据是由用户输入的字符串；提示字符串告诉用户，要输入的信息是什么；默认值则可以用来提示用户用哪种格式输入信息。

例 6_8：显示 3 种对话框。

```
<html>
<head>
  <script type="text/javascript">
        function fun1()
        {
            ans=confirm("你选择的是 confirm 对话框吗？");
            if(ans==true)
              alert("恭喜你！答对了！");
            else
              alert("错!你选择的是 confirm 对话框！");
        }
        function fun2()
        {
          ans=prompt("你学过 Java 语言吗？","我学过了！");
          alert("你的回答是"+ans);
        }
  </script>
</head>
<body>
<center>
  <font color="blue" size="20">对话框的显示</font>
  <form action="" name="frm" >
      <input type="button" name="alertDlg" value="警告对话框" onclick=alert("这是一个警告对话框")>
      <input type="button" name="confirmDlg" value="确认对话框" onclick="fun1()">
      <input type="button" name="alertDlg" value="提示输入对话框" onclick="fun2()">
  </form>
</center>
</body>
</html>
```

6.3.3　文档对象

文档对象也是窗口对象的下一层对象。凡是在浏览器中看到的全部内容，除了工具栏和状态栏，都算文档对象的范围。虽然文档对象是窗口对象下多个对象的其中一个，但是这个对象却是最重要的一个对象，它包含了所有在浏览器中看到的对象。文档对象只有属性和方法，而没有事件，其属性大部分控制的就是用户在浏览器上所看到的效果。

1．文档对象的常用属性

forms：是所有窗口的一个数组，forms[0]为第一个窗口，以此类推。

document.title：在程序中设置浏览器窗口的标题。例如，document.title="Welcome"。

bgColor：设置此 HTML 文件的背景颜色，此属性可以读，也可以修改。读时，返回的值是“#”加上 6 个十六进制数；修改时，可以直接输入颜色的名字（red，blue 等）或其颜色的数值（#ffffff，#df343a，#000000 等）。例如，document.bgColor="blue"。

location：是当前正在被浏览的网页的位置，也就是地址栏中的位置。此属性为只读。

2．文档对象的常用方法

write()：将文字输出到浏览器的窗口中。

writeln()：将文字输出到浏览器的窗口中，末尾加一个回车换行。

open([文件格式])：这个方法用于新文档的创建。是在一个 HTML 文件中打开另一个 HTML 文件，就好像按下超链接一样，只是其中的内容并不是一个已经存在的文件，而是利用 open()后的 write()或 writeln()所动态产生的。参数文件格式是可选项，告诉浏览器其后的文件数据是哪一种格式。例如，document.open("text/html")。这个方法与窗口对象的 open()方法不同，请注意区别。

close()：将 open()打开的数据流关闭。若在 document.close()后还有 document.write()或 document.writeln()，则浏览器只显示 close()后面的内容。例如：

```
document.open();
    document.write ("HELLO");
    document.close();
    document.write("Everyone") ;
```

浏览器只显示“Everyone”，“HELLO”不显示。

clear()：和 close()方法不同的是，它会把它前面和后面的内容一起显示。例如：

```
document.open();
    document.write ("HELLO");
    document.clear();
    document.write("Everyone");
```

浏览器显示：“HELLO Everyone”。

例 6_9：文档对象的应用。

```
<html>
<head>
  <script>
      function fun1()
      {
          document.title="  文档对象的应用";
          document.bgColor="yellow";
      }
      function fun2()
      {
          document.open();
          document.write ("HELLO " ) ;
        document.close();
        document.write("Everyone") ;
      }
```

```
    </script>
</head>
<body>
    <form name="frm" action="">
        <input type="button" value="改变背景颜色" name="colorBtn" onclick="fun1()">
        <input type= "button" value= "显示地址" name= "locationBtn" onclick=alert(document.location)>
        <input type="button" value="写数据" name="btn" onclick="fun2()">
    </form>
</body>
</html>
```

6.3.4　表单对象

表单对象是文档对象的一个子对象，表示在文档中定义表单。页面上的各种控件实际上都是被放在表单里的。例如，在创建了一个表单对象后，就可以利用<input type="text">创建文本框对象。

1．表单对象的引用

可以采用两种方法来引用表单对象：

第 1 种方法是直接采用表单对象的 name 属性。例如，<form name="form1">则可以通过 document.form1 来引用表单对象，或直接通过 form1 来访问该表单。

第 2 种方法是通过 forms 数组。数组的下标可以使用表单的名字或表单在程序中出现的顺序索引值。注意：下标是从 0 开始的，forms[0]代表在程序中定义的第一个表单。例如：

```
<form name= "form1">
</form>
<form name= "form2">
</form>
```

由上可知，document.forms[0]和 document.forms[form1]都表示在程序中定义的第 1 个表单 form1，document.forms[1]和 document.forms[form2]都表示在程序中定义的第 2 个表单 form2。

2．表单对象的常用属性

name：表示表单的名字，这个属性可以读取也可以修改，但是一般不修改表单的名字。

action：通常在设计完一个表单后，如果想把用户所填的数据记录在服务器上，或是执行 JavaScript 做不到的功能，就必须将数据送到服务器上让 JSP 去执行，action 属性就是记录这个程序的路径及文件名。

method：就是把信息传递给 action 属性所指定的程序所使用的方法。一共有两种方法，一种是 get 方法，一种是 post 方法。

elements：在表单中会有多个对象，这多个对象会被放到一个数组中，就是 elements 数组。数组的下标由每个对象在该表单中出现的顺序决定，表单中第一个对象的下标是 0。例如：

```
<form name="frm">
  <input type="text" name="txt" >
  <input type= "button"name="btn"value= "ok">
</form>
```

由上可知，frm.elements[0]指的是文本框，frm.element[1]指的是“ok”按钮。一般不采用这种方法来指明对象，因为这样程序的可读性比较差。

3．表单对象的方法

表单对象中有两个方法：一个是 submit()，另一个是 reset()。submit()方法的功能是将用户在窗口中所填写的内容送给服务器，也就是提交。这一般被用于用户输入数据后就直接送出，而不需要用户按下按钮送出的情况。reset()方法的功能是重置，使窗口中的输入值恢复到原来尚未输入时的状态。

例 6_10：表单的应用。

```
<html >
<head>
   <title>Javascript 中表单的应用</title>
</head>
<body>
  <form id="form1" name="form1" method="post" action=" ">
       <input type="text" name="txt1" value="表单 1 的内容" >
     <input type="submit" name="one" value="表单 1 提交" />
  </form>
  <form id="form2" name="form2" method="post" action=" ">
       <input type="text" name="txt2" value="表单 2 的内容">
     <input type="submit" name="two" value="表单 2 提交" />
  </form>
  <form id="form3" name="form3" action="" >
  <table width="675" border="2" cellspacing="5" cellpadding="3">
  <tr>
    <td width="142">该页包含表单数量：</td>
    <td width="258"><script>document.write(document.forms.length)</script></td>
    <td></td>
    <td></td>
  </tr>
  <tr>
    <td>第一个表单中按钮值是：</td>
    <td><script>document.write(window.document.forms[0].one.value)</script></td>
    <td>第一个表单中文本框中的值是：  </td>
    <td><script>document.write(window.document.forms[0].txt1.value)</script></td>
  </tr>
  <tr>
```

```
        <td>第二个表单中按钮值是：</td>
        <td><script>document.write(document.forms[1].two.value)</script></td>
        <td>第二个表单中文本框中的值是： </td>
        <td><script>document.write(window.document.forms[1].txt2.value)</script></td>
      </tr>
      </table>
      </form>
    </body>
    </html>
```

6.3.5 历史对象

在浏览网页时，经常会做的一个动作就是按 “上一页”按钮或 “下一页”按钮来查看已经看过的内容或没有看的内容。而这些网页的链接，全部合起来，就可以被看做是一个历史对象。上过网的人都知道，这种存储网页路径的功能，仅限于一次性打开浏览器的动作；如果关掉浏览器的窗口再重新打开或另外打开一个新的窗口，这些原本在历史对象中的浏览记录就会消失。

而历史对象的主要功能就是存储这些链接的资料，并提供给用户一些方法以直接跳到某一个已经浏览过的网页页面，而不必为了要回到几个之前的网页而辛苦地按“上一页”按钮。

历史对象是根据浏览过的网页的先后顺序来记录网页的。需要注意的是，历史对象记录下来的网页都必须是使用超链接方式的，也就是说，如果在地址栏中输入网址来打开另一个网页，历史对象是不会记录该网页的。

历史只含有属性和方法，没有事件。

历史对象的属性共有 4 个。它们分别是：length，current，next，previous。

length：表示当前历史对象共存储了几个网页，返回一个正整数，为只读属性。

current：表示当前正在浏览的网页的位置。

previous 和 next 表示上一页和下一页的网页的位置。

历史对象共有 3 个方法。它们分别是：back()，forward()，go()。其功能都是让用户可以跳转到其他网页。

back()：此功能和按下“上一页”按钮一样，是回到上一个页面。

forward()：此功能和按下“下一页”按钮一样，是到下一个页面。

go(页面个数)：back()方法和 forward()方法都只能跳转一个页面，如果想要跳转多个页面，就需要用 go()方法。参数是一个整数，若是正整数 n，表示要跳转到下 n 个页面；若是负整数 n，表示要回到上 n 个页面。例如：

```
history.go(-1)；//回到上一页，等同于 back()
history.go(0)；//页面不动，重新读取
history.go(1)；//回到下一页，等同于 forward()
```

例 6_11：历史对象的应用。

```
<html>
<head>
<script src="text/javascript">
    function fun()
    {
      n=frm.txt.value;
      history.go(n);
    }
</script>
</head>
<body>
<script>
  document.write("你已经访问了"+history.length+"个页面");
</script>
      <form action="" name="frm">
            <input type="button" name="btn1" value="下一页>" onclick="history.forward()">
            <input type="button" name="btn2" value="上一页" onclick="history.back()">
            <input type="button" name="btn3" value="跳转多页" onclick="fun()">
            <input type="text" size=5 name="txt"> 页
       </form>
</body>
</html>
```

6.4　个人用户注册页面的实现

6.4.1　验证方法

个人用户注册开发一般分两步完成。第 1 步是写注册表单，注册表单决定了用户注册时需要填写的信息；第 2 步是写注册表单提交页面，在这个页面完成注册验证和提交给数据库的动作。在用户填写注册表单时，需要对填写的信息进行合法性验证，例如，如果用户输入的邮箱地址没有“@”,则需要用户重新输入等。这些都需要在提交注册表单以前用 JavaScript 进行验证。

如何在提交表单以前用 JavaScript 对用户的输入信息进行合法性验证？可以采用两种方法。第 1 种方法是创建一个普通按钮，设置 onclick 属性，利用函数对输入信息进行验证，然后写代码来提交。第 2 种方法是对提交按钮编程，但由于提交按钮在默认情况下不管检查结果如何都会提交表单，所以需要在开始时设置表单的属性 onSubmit 为 false，在通过检查后，再设置其值为 true，这时不再需要调用表单的 submit()方法。

例 6_12：用户输入年龄，由于年龄只能是整数，对其进行合法性验证。

```
<html>
```

```
<head>
  <script>
      function fun1()
      {
          var age=frm.ageTxt.value;
          len=age.length;
        if(len>=3)
        {
            alert("年龄只能是小于 100 的数，请重新输入！");
            frm.ageTxt.value='0';
        }
        else
        {
          for(i=0;i<=len-1;i++)
          {
            s=age.charAt(i);
            if(s<='0'||s>='9')
            {
                    alert("输入只能是整数，请重新输入！");
                    frm.ageTxt.value='0';
                    break;
             }

          }
          if(i==len)
          {
            alert("输入正确，可以提交！");
            form.submit();
          }
        }
      }
  </script>
</head>
<body>
<form action="" name="frm">
    请输入您的年龄：
    <input type="text" name="ageTxt" value="0" >
      <input type="button" name="ok" value="提交" onclick="fun1()">
</form>
</body>
</html>
```

6.4.2　实训：仿照上例，完成注册页面的验证

个人用户注册页面是进行个人注册。注册信息包括用户名、密码、密码确认、真实姓名、年龄、性别、出生日期、电子邮箱、备注信息等。首先需要用 HTML 完成注册页面的显示，然后用 JavaScript 对注册信息进行验证。

需验证的信息包括：判断用户是否输入信息；真实姓名只能是汉字和英文字符，不能使符号等特殊字符；出生日期中，年只能是 1900～2010 之间的数字，月只能是 1～12 之间的数字，日只能是 1～31 之间的数字；年龄必须是整数；输入电子邮箱时必须含有“@”和“.”符号。需要定义按钮的单击事件处理函数完成验证。

6.5　JavaScript在项目中的应用

前面讲解了 JavaScript 的基本概念及用法，现结合人力资源项目中两个模块的例子来讲解 JavaScript 在项目中的具体应用。

6.5.1　登录实现

为了能够正常登录，在单击“登录”按钮时需要在客户端对输入数据用户名及密码进行不为空的验证，避免将数据传入后台服务器验证，从而提高代码的执行效率。下边就结合具体例子详细讲解客户端是如何进行表单数据验证的。

登录页面 login.jsp 如图 6.7 所示。

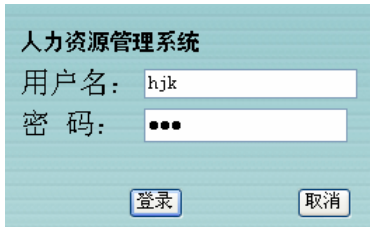


图 6.7　人力资源管理系统登录页面

登录页面 login.jsp 的主要源代码如下。

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>My JSP 'login.jsp' starting page</title>
A.  <script language="javascript">
B.  function check(){
            var account=document.login.useraccount.value;
            var pwd=document.login.userpwd.value;
            if(account==""){
```

```
                    alert("请输入账号!");
                    document.login.useraccount.focus();
                    return false;
                }

                if(pwd==""){
                    alert("请输入密码!");
                    document.login.userpwd.focus();
                    return false;
                }
                document.login.action="../../bank/UserLoginAction";
            }
        </script>
        </head>
        <body>
            <div>
                <image style="height:15%;width:40%;" src="../image/loginlogo.gif" />

            </div>
            <div style="margin-top: 7%; margin-left: 29.5%; margin-right: ;">
                <form name="login" method="post">
                    <table>
                        <tr>
                            <td align="right">
                                <font face="黑体" size="1">人力资源管理系统</font>
                            </td>
                        </tr>
                    </table>

                    <font color="red"> <%
        String state = (String) session.getValue("state");
        if (state == null)
            state = "";
    %> <%=state%> </font>
                    <table>
                        <tr>
                            <td>
                                用户名:
                            </td>
                            <td>
            C.              <input type="text" name="useraccount">
                            </td>
                        </tr>
                        <tr>
```

```
                                    <td>
                                        密 码:
                                    </td>
                                    <td>
            D.  <input type="password" name="useupwd">
                                    </td>
                                </tr>
                            </table>
                            <br>
                            <table>
                                <tr>
                                    <td>
                                              
        E.  <input type="submit" value="登录" onclick="return check()">
                                    </td>
                                    <td>
                                              
                <input type="reset" value="取消">
                                    </td>
                                </tr>
                            </table>
                        </form>
                    </div>
                </body>
            </html>
```

在单击“登录”按钮时，需要将表单数据提交后台 Servlet 进行处理，但在数据被提交到 Servlet 之前在客户端需要对输入数据进行验证。若将数据直接提交到后台进行验证会影响程序的执行效率。那么如何在客户端对表单数据进行验证呢？首先需要在<body>…</body>之前，在<head>…</head>内插入 JavaScript 代码，如源代码中标记 A 处所示。它以<script language="javascript">开始，以</script>结束。在它们之间定义一个方法 check()实现表单数据验证功能，如标记 B 处所示。该方法名称可以随便取，但需要遵守 Java 函数命名规则。

在方法 check()中定义了两个变量 account 和 pwd，然后通过文档对象调用窗口名称 login。它是表单对象的 name 属性值，如<form name="login" method="post">。再将窗体对象 login 属性 useraccount 和 userpwd 的值分别赋给变量 account 和 pwd。而属性 useraccount 和 userpwd 是表单里两个输入控件的 name 属性值，如标记 C 和 D 处所示。这样在方法 check()中就可以调用控件所输入的值，对得到的两个变量值进行是否为空的判断。如果为空则调用弹出警告对话框方法 alert()以提示用户。若输入值为空，则方法返回 false,表示禁止提交数据。如果两个变量都不为空，则最后执行语句“document.login.action="../../bank/UserLoginAction";”，表示执行 UserLoginAction 这个 Servlet，并将表单数据用户名和密码传入后台 UserLoginAction。

当单击“登录”按钮时，程序又怎么自动去调用 JavaScript 里的 check()方法呢？标记

E 处可以知道登录按钮输入框里有一个属性 onclick，表示单击鼠标时，将执行它所对应的值，而它的值就是 JavaScript 方法 check()。这样在表单数据还未被提交到服务器端便完成了数据的验证，从而提高开发效率。

为了便于大家更加深刻理解 JavaScript 的用法，下面再结合人力资源项目中用户管理模块的例子来讲解它的用法。

6.5.2　用户管理实现

通过用户管理模块，主要是可以通过支行名、用户组名、用户账号进行查询，可以进行修改系统用户信息、修改权限、删除该用户等操作。在进行删除该用户操作时需要在客户端提示用户是否需要删除，这就需要 JavaScript 的验证。用户管理页面如图 6.8 所示。

图 6.8　用户管理页面

用户管理页面 userMange.jsp 的源代码如下。

```
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=gb18030"
pageEncoding="GB18030"
%>
<%@ page import="java.util.ArrayList"%>
<%@ page import="com.etop.bank.usermanage.bean.UserAccountBean"%>
</style>
<TITLE>userMaIndex.jsp</TITLE>
<script language="javascript">
function adduser(){
URL="../../bank/userManager/addUsers.jsp";
window.location=URL;
}
```

```
function deleteusers(){
if(!confirm("要删除选中的用户吗？ ")){
return false;
}
window.location="../../bank/UserAccountAction?option=deletebyids";
}


function deleteById(id){
if(!confirm("要删除该用户吗?")){
return false;
}}

function deleteChecked(){
  if(!confirm("要删除选中的用户吗?")){
      return false;
  }

}
function changeThegroupIdV(){
      for(var i=0;i<document.users.groupidS.options.length;i++){
        if(document.users.groupidS.options[i].selected){
      document.users.groupid.value=document.users.groupidS.options[i].value;
        }
   }
}
function changeTheSubankIdV(){
    for(var i=0;i<document.users.subbankidS.options.length;i++){
        if(document.users.subbankidS.options[i].selected){
                document.users.subbankid.value=document.users.subbankidS.options[i].value;
        }
   }
}
</script>
</HEAD>
<BODY>
当前位置：用户管理
<BR>
<BR>
<%ArrayList accounts=(ArrayList)request.getAttribute("Alluseraccount");
  UserAccountBean account=null;
%>
<FORM name="users" method="post">
```

```
        <input type="hidden" name="subbankid" value="">
        <input type="hidden" name="groupid" value="">

        <TABLE cellspacing="2" cellpadding="2" border="0" width="70%" height="34">
        <TR>
                <TD width="35%">支 行</TD>
                <TD>
                    <SELECT name="subbankidS" onclick="changeTheSubankIdV()">
                    <% for(int i=0;i<accounts.size();i++){
                            account=(UserAccountBean)accounts.get(i);
                     %>
                     <OPTION value="<%=account.getSubbankid() %>"><%=account.getSubbankid()
%></OPTION>
                     <%}%>
                    </SELECT></TD>
                <TD width="35%">用户组</TD>
                <TD><SELECT name="groupidS" onclick="changeThegroupIdV()">
                <%for(int i=0;i<accounts.size();i++){
                     account=(UserAccountBean)accounts.get(i);
                %>
                     <OPTION
value="<%=account.getGroupid()%>"><%=account.getGroupid()%></OPTION>
                <%}%>
                </SELECT></TD>
                <TD width="60%">
                用户账号
                </TD>
                <TD><INPUT type="text" name="useraccount" size="10"></TD>
                <TD><INPUT type="submit" value="查 找 "
onClick="javascript:document.users.action='../../bank/useraccountAction?option=selectaccountbykey'"></TD>
                <TD><INPUT type="button" value="全 部 "
onClick="javascript:window.location='../../bank/useraccountAction?option=selectallaccount'"></TD>
            </TR>
        </TABLE>

        <BR>

        <TABLE>
            <TR>
                <TD><INPUT type="button" value="添加用户" onClick="adduser()"></TD>
                <TD width="48"></TD>
                <TD width="71"><INPUT type="button" value="删除选中用户" onClick="return
deleteusers()"></TD>
```

```
            </TR>
        </TABLE>

        <BR>

        <TABLE  border="1">
        <TBODY>
                <TR >
                    <TD  background="image/th.gif"  width="1%"  ><INPUT  type="checkbox"
value="*"></TD>
                    <TD background="image/th.gif"  width="11%" >用户名</TD>

                    <TD background="image/th.gif"  width="11%" >用户组</TD>
                    <TD background="image/th.gif"  width="11%" >部门</TD>
                    <TD background="image/th.gif"  width="11%" >职位</TD>
                    <TD background="image/th.gif"  width="11%" >支行名</TD>
                    <TD background="image/th.gif"  width="11%" >角色</TD>
                    <TD background="image/th.gif"  width="20%">操作</TD>
                </TR>

                <% for(int i=0;i<accounts.size();i++){
                        account=(UserAccountBean)accounts.get(i);
                %>
                    <TR>
                        <TD                  ><INPUT  type="checkbox"  name="ids"
value="<%=account.getId()%>"></TD>
                          <TD  ><%=account.getUseraccount()%></TD>

                        <TD  ><%=account.getGroupid()%></TD>
                        <TD  ><%=account.getDepartmentid() %></TD>
                        <TD  ><%=account.getPositionid() %></TD>
                        <TD  ><%=account.getSubbankid() %></TD>
                        <TD ><%=account.getUsertype()%></TD>
                        <TD  >
                    <A
href="../../bank/userManager/updateUsersInformation.jsp?id=<%=account.getId()%>">【修改系统用户信息】</A>
                    <BR>
                    <A
href="../../bank/userManager/updateRight.jsp?id=<%=account.getId()%>&group_id=<%=account.getGroupid()%>">
【修改权限】</A>
                  <BR>
                  <A  href='../../bank/UserAccountAction?option=deletebyid&id=<%=account.getId()%>'
                   onclick="  return  deleteById(<%=account.getId()%>)"> 【 删 除 该 用 户 】 </A>
</TD>
```

```
                    </TR>
                    <%} %>
                </TBODY>
            </TABLE>
            </FORM>
            </BODY>
            </HTML>
```

在本例子中，删除用户、删除几个选中用户、添加用户、选中下拉列表数据时获取下拉列表值等功能都是通过调用 JavaScript 语句实现的。现分别对它们进行一一介绍。

（1）实现删除用户功能的 HTML 语句如下。

```
<A href='../../bank/UserAccountAction?option=deletebyid&id=<%=account.getId()%>' onclick=" return deleteById(<%=account.getId()%>)">【删除该用户】</A>
```

这时将执行 JavaScript 里的方法 deleteById(id)以确定是否删除该用户，弹出确认对话框以提示用户操作，要删除时单击“确定”按钮，否则单击“取消”按钮。代码如下。

```
function deleteById(id){
if(!confirm("要删除该用户吗?")){
return false;
}}
```

删除该用户的确认对话框如图 6.9 所示。

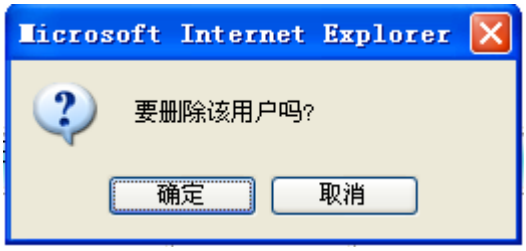


图 6.9　删除该用户的确认对话框

（2）实现删除几个选中用户功能的 HTML 语句如下。

```
<TD width="71"><INPUT type="button" value="删除选中用户" onClick="return deleteusers()"></TD>
```

这时将执行 JavaScript 里的方法 deleteusers()以确定是否删除选中用户，弹出确认对话框以提示用户操作，要删除时单击“确定”按钮，否则单击“取消”按钮。代码如下。

```
function deleteusers(){
if(!confirm("要删除选中的用户吗？")){
return false;
}
window.location="../../bank/UserAccountAction?option=deletebyids";
}
```

删除选中用户的确认对话框如图 6.10 所示。

图 6.10　删除选中用户的确认对话框

（3）实现添加用户功能的 HTML 语句如下。

```
<TD><INPUT type="button" value="添加用户" onClick="adduser()"></TD>
```

这时将执行 JavaScript 里的方法 adduser()。代码如下。

```
function adduser(){
URL="../../bank/userManager/addUsers.jsp";
window.location=url;
}
```

将添加用户页面 addUsers.jsp 赋给 URL，再通过窗口对象的 location 加载 URL，从而弹出添加用户页面。该页面如图 6.11 所示。

图 6.11　添加用户页面

（4）选中支行下拉列表数据时获取下拉列表值的 HTML 语句如下。

```
<SELECT name="subbankidS" onclick="changeTheSubankIdV()">
```

这时将执行 JavaScript 里的方法 changeTheSubankIdV()，表示当选中支行下拉列表中的一个数据时，将给变量 subbankid 赋值。代码如下。

```
function changeTheSubankIdV(){
    for(var i=0;i<document.users.subbankidS.options.length;i++){
        if(document.users.subbankidS.options[i].selected){
                document.users.subbankid.value=document.users.subbankidS.options[i].value;
        }
    }
```

（5）选中用户组下拉列表数据时获取下拉列表值的 HTML 语句如下。

```
<TD><SELECT name="groupidS" onclick="changeThegroupIdV()">
```

这时将执行 JavaScript 里的方法 changeThegroupIdV()，表示当选中用户组下拉列表中的一个数据时，将给变量 groupid 赋值。代码如下。

```
function changeThegroupIdV(){
    for(var i=0;i<document.users.groupidS.options.length;i++){
        if(document.users.groupidS.options[i].selected){
                document.users.groupid.value=document.users.groupidS.options[i].value;
        }
    }
}
```

6.6　实训操作

在此次实训中，主要是要熟练掌握 JavaScript 的用法。此次实训的例子要实现采用第 5 章的图 5.15 及图 5.16 的功能。两个页面中的所有功能都用 JavaScript 实现。例如，点击“查询”按钮时调用 checkuser()方法，点击“删除用户信息”按钮时调用 delUserById()方法，点击“编辑”按钮时调用 editUserById()方法等。

习　题

一、单项选择题

1．以下有关变量声明的说法中，哪一个是错误的？（　　）

A．变量名不能含有下画线。

B．变量名的第一个字符必须是字母。

C．变量名的字符最多不能超过 255 个。

D．变量名不能使用保留字。

2．表达式 10 %3+2^4\5 的执行结果是（　）。

A．2

B．16

C．4

D．3

3．str"abcdeabc"；　x=str.indexOf("d");执行结果 x 是（　）。

A．3

B．4

C．5

D．6

4．下列程序段的执行结果 c 为（　　）。

```
a=1;
b=2;
c=3;
if(a*b>2){
    c=a+b;
}
```

A．5

B．3

C．1

D．2

5．下列程序段的执行结果 sum 为（　　）。

```
sum=0;
for(i=1;i<=5;i++){
    sum=sum+i*3;
    }
```

A．15

B．30

C．45

D．18

6．下面哪一个不是窗口对象的子对象？（　）

A．document

B．history

C．frame

D．request

7．下面哪一个不是表单对象的子对象？（　　）

A．link

B．radio

C．button

D．reset

8．下面哪一个不是客户端对象？（　　）

A．project

B．document
C．window
D．link
9．下面哪一个不是文档对象的属性？（　　）
A．link
B．fgColor
C．bgColor
D．tlink
10．JavaScript 是运行在（　　）的程序。
A．客户端
B．服务器端
C．客户端，服务器端均可以
D．客户端，服务器端均不可以

二、编程题

1．编写程序，显示当前日期。
2．从文本框中输入两个整数，利用类型转换完成两个数的相加。
3．在文本框中输入年龄，验证其合法性。
4．在文本框中输入邮箱地址，验证其合法性。
5．建立用户登录页面，判断用户名是否是 admin，密码是否是 123456，用户输入完毕后，需要让用户确认是否提交。如果用户名和密码不正确，用对话框提示。

第 7 章　JSP技术

📖 知识点

- JSP 的有关概念和特点
- JSP 的生命周期
- JSP 的基本结构
- JSP 脚本
- JSP 指令
- JSP 动作
- JSP 的内置对象
- JSP 和 Servlet 的相互调用

📣 难点

- JSP 对表单的处理
- JSP 指令和 JSP 动作
- JSP 的内置对象

✍掌握

- JSP 的概念、特性和机制
- JSP 脚本、指令和动作
- JSP 的内置对象
- JSP 和 Servlet 的相互调用

任务引入

在 Java Web 项目开发中，比较经典的开发模式是 MVC 模式。M 表示 Model（模型），V 表示 View（视图），C 表示控制（Control）。其中，视图层可以用 HTML 或 JSP 实现，主要用于显示。HTML 的用法在前面已经讲过，本章主要讲解 JSP 的概念及用法。

7.1　JSP简介

JSP 是 Java Server Pages 的缩写，是 Sun 公司于 1999 年推出的动态网页技术。利用这种技术可以建立先进、安全、跨平台的动态网站。它将常规的静态 HTML 和动态产生的内容相结合，具有 Java 技术的“编写一次，随处运行”的特性，使网页的外观设计与隐藏在其后的实现逻辑截然分开，提高了开发效率。其主要特点是在传统的 HTML 网页中加入 Java 程序片断和使用各种各样的 JSP 标签，构成 JSP 网页。Web 服务器在接收到客户端访问 JSP 网页的请求时，首先执行其中的程序片段，然后将执行结果以 HTML 格式返回给客户端。嵌入 HTML 网页的程序片断可以完成操作数据库、重定向网页、文件上传等功能，而且所有程序的操作都在服务器端执行，在网络上传送给客户端的仅仅是程序运行后得到的结果。通过这种方式，极大地减轻了网络的负担。另外，JSP 作为 J2EE 的 Web 服务器组件，可

以通过 JSP 网页以多种方式同企业系统进行交互。JSP 具有简单易用及可移植等优点，这使它成为目前最流行的动态网页设计技术之一。

7.1.1　一个简单的JSP程序

例 7_1：显示当前日期。

```
<html>
  <head>
      <title>My JSP 'First.jsp' starting page</title>
  </head>
  <body>
    This is my JSP page.
      <% java.util.Date d=new java.util.Date(); %>
      Today's date is<%= d.getDate() %> and this JSP page worked!
    <br>
  </body>
</html>
```

这是一个简单的 JSP 程序。页面中有普通的、静态的 HTML，如<title>My JSP 'First.jsp' starting page</title>。“This is my JSP page.”这句话会显示在 Web 页面上，紧接着 JSP 会产生一个动态的部分，显示当前日期。JSP 就是这样将静态的 HTML 和动态产生的内容结合起来的。

在传统的 HTML 页面中加入 Java 程序片断和 JSP 标签就构成了一个 JSP 页面。也就是说，一个 JSP 页面除了具有普通的 HTML 标记符外，还有标记符号“<%…%>”以加入 Java 代码。JSP 文件的扩展名是 jsp，文件的命名规则必须符合标识符的规定，即由字母、数字、下画线和美元符号“$”组成，第一个不能是数字。需要注意的是，JSP 是基于 Java 的，名字需要区分大小写。

当多个客户端请求一个 JSP 页面时，JSP 会为每个客户端启动一个线程，而不是启动一个进程。这样可以提高效率。

程序的运行结果如图 7.1 所示。

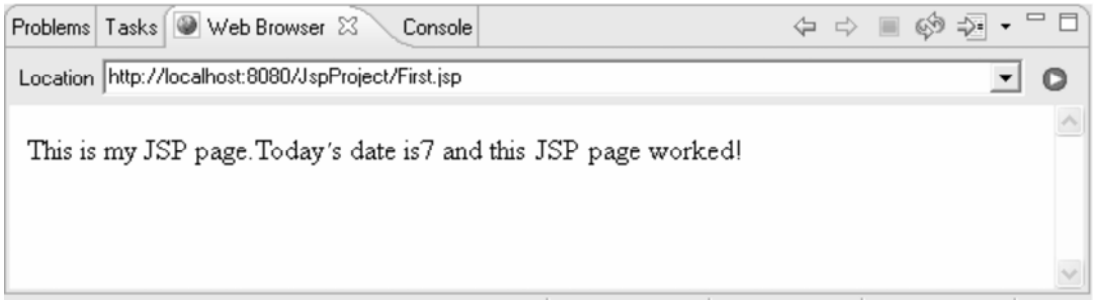


图 7.1　一个简单的 JSP 程序的运行结果

7.1.2　运行方式

JSP 的运行方式和其他动态网页技术有所不同。从形式上看，它是嵌入网页的脚本，脚本可以完成各种不同的功能，而返回给客户端的仅仅是一个 HTML 页面。当 Web 浏览器发出请求，请求查询 JSP 页面时，在服务器端，JSP 解析器会解析 JSP 源文件的内容，根据 JSP 源文件的内容创建临时的 Servlet 源代码，这些 Servlet 除了创建页面的动态元素，还负责显示在原 JSP 页面上定义的静态元素。接着 Java 编译器将 Servlet 编译生成 class 文件，Servlet 实例化调用 Servlet 的 init()方法，然后调用 service()方法执行 Servlet 逻辑。最后 Web 服务器将静态的 HTML 页面和图像与在 JSP 页面中声明的动态元素相结合，通过 Servlet 生成动态的 HTML 页面传送给 Web 浏览器。例如，对于例 7_1 中的 JSP 程序，可以在 %TOMCAT_HOME%\work\Catalina\localhost\test\org\ apache\ jsp\ 下找到生成的 Servlet 程序的 First.java，该程序经编译生成 First.class 程序，在 %TOMCAT_HOME%\work\Catalina\localhost\test\org\ apache\ jsp\ 文件夹下。JSP 的运行过程如图 7.2 所示。在第一次访问 JSP 页面时，由于要先对其进行编译，所以运行速度较慢。但以后访问 JSP 页面时，直接调用 Servlet，由于 Serlvet 运行速度很快，所以在这种情况下访问速度就非常快。

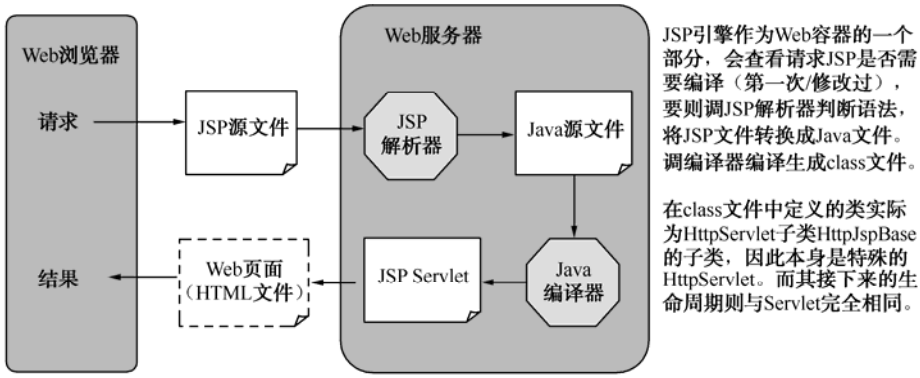


图 7.2　JSP 的运行过程

7.1.3　JSP的生命周期

在 JSP 文件生成 Servlet 后，其生命周期与 Servlet 相同，只是调用方法名称不同。JSP 的生命周期分为编译、装载、初始化、执行和销毁 5 个阶段。

当浏览器请求访问一个 JSP 页面时，JSP 引擎首先查看它是否需要编译。如果页面从来没有编译过，或 JSP 页面自上次编译之后又经过修改，那么 JSP 引擎就会编译该页面。编译时，首先解析 JSP 页面，没有语法错误后，将 JSP 转换成一个 Servlet，接着编译 Servlet。

然后进行装载工作，由于 JSP 在实际装载之前，需要转换成一个 Servlet，因此 JSP 的装载过程和 Servlet 的装载过程非常相似。其主要区别在于 JSP 的类文件往往是重装载的，即使同名的类文件已被装载到内容中也需要重装载。不过，尽管 JSP 类已经被重新装载了，任何其他相关的类，如 JavaBean 或公共类，也不总是自动被装载的。一些 JSP 引擎可能自

动装载其他类，但是不能完全指望它们自动装载所有需要的类。

第三步进行初始化工作。由于 JSP 已经被编译成 Servlet，因此 JSP 的初始化也就是 Servlet 的初始化。如果要执行 JSP 特定的初始化，可以重写_jspInit()方法。

JSP 引擎通过调用 JSP 的 jspService(HttpServletRequest req,HttpServletResponse resp)方法来执行用户发出的请求。注意不能重写这个方法。

JSP 的销毁工作调用的是_jspDestroy()方法，其对应 Servlet 的 destroy()方法。如果要执行像释放数据库连接和关闭已打开文件等的清理工作，就应该使用 jspDestroy()方法。

7.1.4　处理汉字信息

当用 Servlet 或 JSP 编写程序，对表单提交的数据进行显示时，若提交的数据是汉字时，往往会出现乱码现象。这是由于页面显示采取的编码方式是“GB2312”，而 Tomcat 服务器采用的编码方式是“ISO-8859-1”，编码方式的不同导致了乱码的出现。所以对含有汉字的信息必须进行特殊的处理。首先，将获取的字符串用“ISO-8859-1”进行编码，并将编码存放到一个字节数组中，然后再将这个数组转换成字符串。代码如下。

```
String str=request.getParameter("参数名称");
byte b[]=str.getBytes("ISO-8859-1");
str=new String(b);
```

通过上述过程，提交的任何信息，无论是汉字还是英文，都可以正确显示，不会再有乱码现象出现。

7.2　JSP 元素

JSP 元素包括 JSP 脚本、JSP 指令和 JSP 动作。JSP 脚本允许将 Java 代码插入到将要在 JSP 页面中生成的 Servlet 中。JSP 指令是在 JSP 页面被编译成 Servlet 时由 JSP 引擎处理的指令。JSP 页面指令传递和页面相关的信息，包含指令可以被用来包含一个外部文件。JSP 动作是在 JSP 页面中使用 XML 语法的一种定义。JSP 动作包含使用 JavaBean 组件执行用户输入验证、处理数据库访问、动态插入文件、把用户转发给其他页面等。

7.2.1　JSP脚本

JSP 脚本一共有 3 种类型。除了在例 7_1 中使用过的“<%…%>”外，还有声明变量、方法和类所用的“<%!…%>”和显示表达式的值所用的“<%= …%>”等。

1．声明变量、方法和类

可以在<%!…%>内声明 Java 中允许的任何类型的变量和方法，也可以声明一个类。声明变量、方法和类的格式和 Java 中的声明格式相同。在 JSP 中只需要把声明放在<%!…%>内即可。

在<%!…%>内声明的变量、方法和类在整个 JSP 页面中都是有效的。当多个用户请求

同一个 JSP 页面时，将共享这些声明的变量、方法和类，如果一个用户修改了变量的值，那么其他用户看到的也将是修改后的值。

需要注意的是，“<”、“%”和“!”之间都没有空格。

例 7_2：编写 JSP 程序，求 1 到 100 的和。

Ex7_2.html

```
<html>
  <head>
    <title>Ex7_2.html</title>
  </head>
  <body>
    <FORM action="Ex7_2.jsp" method="post">
      请输入数字：<INPUT type="text" name="num" ><br>
      <p>
      <INPUT type="submit" name="submit" value="提交">
      <INPUT type="reset" name="reset" value="重置">
    </FORM>
  </body>
</html>
```

Ex7_2.jsp

```
<html>
  <head>
    <title>My JSP 'Ex7_2.jsp' starting page</title>
  </head>
  <body>
    <%! int sum=0,s,num;
        String str;
    %>
    <%! int count(int n)
        {
            int i;
            sum=0;
            for(i=1;i<=n;i++)
                sum=sum+i;
            return sum;
        }
    %>
    <%
        str=request.getParameter("num");
        num=Integer.parseInt(str);
        s=count(num);
```

```
                out.println("1 到"+num+"的和为："+s);
            %>
          <br>
        </body>
      </html>
```

在 Ex7_2.html 中定义了一个表单，用于输入一个计算和值的数字，该表单以 post 方法提交。在 Ex7_2.jsp 中计算和值，在程序中定义 4 个变量和一个 count()方法。它们都需要被放在<%!…%>内。

注意：

（1）文本框中的输入的数字实际上是字符串类型，所以在 JSP 文件中需要对其进行类型转换。

（2）变量 i 的定义含义和其他变量 sum，s，num，str 的定义含义是不同的。在<%!…%>内声明的变量和方法在整个 JSP 页面中都是有效的；而变量 i 的定义被放在方法 count()的内部，它只在该方法中有效，在该方法外部不能使用。

程序的运行结果如图 7.3 所示。

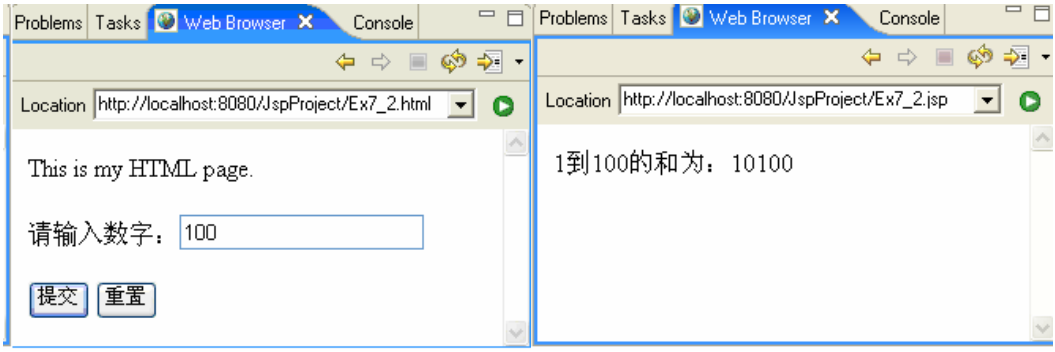


图 7.3　计算和值程序的运行结果

2．插入Java代码

在 JSP 中使用<%　%>标注需要执行的 Java 语句块。<%…%>可以包含任何数量的语句，可以包含变量、方法和类的定义，表达式也可以被放到其中。但要注意，文本、HTML 标记和 JSP 元素必须位于<%…%>之外。需要注意的是，“<”和“%”之间及“%”和“>”之间不能有空格。

例 7_3：根据当前时间显示“早上好”或“下午好”。

```
<html>
  <head>
    <title>My JSP 'Ex7_3.jsp' starting page</title>
  </head>
  <body>
    This is my JSP page. <br>
```

```
        <%
            if(Calendar.getInstance().get(Calendar.AM_PM)==Calendar.AM)
            {%>
                How are you this morning? 早上好!
            <%}
             else
             { %>
                How are you this afternoon? 下午好!
            <% }
        %>
    </body>
</html>
```

该程序根据当前时间输出字符串。需要注意的是，“How are you this morning ?早上好!”和“How are you this afternoon？下午好!”为文本，不能被放在<%…%>内。

3．显示表达式的值

采用<%= 表达式%>格式显示表达式的值。注意：表达式后没有分号，它不能是一条语句。这个表达式必须能求出值，并且将表达式的值以字符串的形式传送给浏览器，如果表达式结果不能被成功转化成字符串，则在请求时抛出一个 classCastException 异常。需要注意的是，“<”、“%”和“=”之间不允许有空格。

例 7_4：显示用户输入的用户名和密码。

Ex7_4.html

```
<html>
    <head>
        <title>Ex7_4.html</title>
</head>
    <body>
        <FORM action="Ex7_4.jsp" method="get">
        用户名：<INPUT type="text" name="sname" >
        密码：<INPUT type="password" name="pwd">
        <p>
        <INPUT type="submit" name="submit" value="提交">
        <INPUT type="reset" name="reset" value="重置">
        </FORM>
    </body>
</html>
```

Ex7_4.jsp

```
<%@ page language="java" contentType="text/html;charset=gb2312"%>
<html>
    <head>
```

```
    <title>My JSP 'Ex7_4.jsp' starting page</title>
  </head>
  <body>
   <%
        response.setContentType("text/html;charset=GB2312");
        String sn=request.getParameter("sname");
        byte b[]=sn.getBytes("ISO-8859-1");
        sn=new String(b);
        String spwd=request.getParameter("pwd");
        byte c[]=spwd.getBytes("ISO-8859-1");
        spwd=new String(c);     %>
    您刚才提交的用户名和密码是：  <br>
    用户名：<%= sn %><br>
    密码：<%= spwd %>
  </body>
</html>
```

该程序被用来显示用户输入的信息。在 JSP 程序中对用户输入的内容进行了处理，以防止出现乱码。<%= sn %>等价于<% out.println(sn); %>。程序的运行结果如图 7.4 所示。

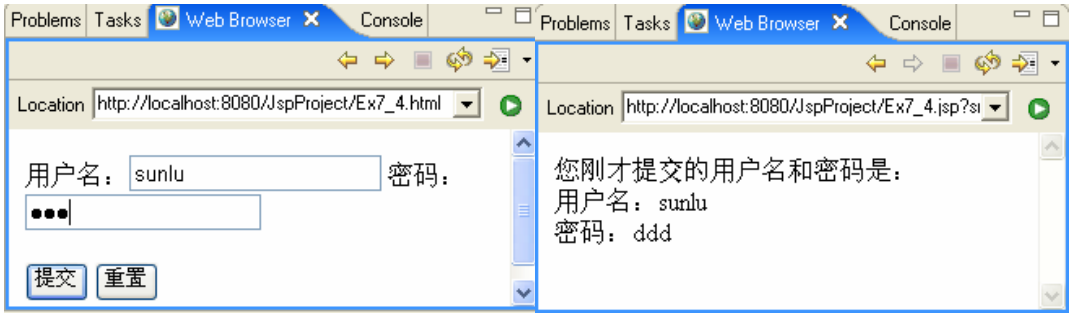


图 7.4　显示用户输入信息程序的运行结果

4．JSP中的注释

在 JSP 中可以用到的注释格式有两种。

（1）HTML 注释。

格式为<!--注释内容- ->。这种注释是要被发送给客户端的，能够出现在浏览器的输出流中。

（2）JSP 注释。

格式为<%-- 注释内容 --%>。这种注释是作为 JSP 本身一部分的隐藏注释，不被发送给客户端。JSP 引擎并不处理这些注释文本。

7.2.2　JSP指令

JSP 指令被用来设置 JSP 页面的各种属性。它可以设置 JSP 页面的字符编码，可以嵌入一个文件，可以导入标签库等。JSP 指令一共有 3 类，包括 page 指令、include 指令和

taglib 指令。

JSP 指令的格式如下。

```
<%@ 指令名 属性 1="属性值 1" 属性 2="属性值 2" 属性 3="属性值 3"…%>
```

需要注意的是，“<”、“%”和“@”之间不允许有空格。

JSP 指令可以被放在页面的任何位置，它对整个页面都是有效的，但通常总是出现在 JSP 文件的开头，定义在其他 JSP 标签的前面。

1．page指令

page 指令用于给页面设置各种属性，JSP 引擎根据这些属性的设置来设置页面。可以导入类，设置 Servlet 的父类，设置内容类型等。它的格式如下。

```
<%@ page 属性 1="属性值 1" 属性 2="属性值 2" 属性 3="属性值 3"…%>
```

各种属性名区分大小写，属性值是用单引号或双引号括起来的。在一个 JSP 页面中可以使用多个 page 指令来设置属性。可以为 import 属性指定多个值，多个值之间用逗号隔开，只能为其他属性指定一个值。例如：

```
<%@ page language= "java" %>
<%@ page import = "java.util.* ", "java.awt.* " %>
<%@ page import= "java.applet.* " %>
```

在该例中，page 只能是小写，各种属性值是用双引号括起来的；一共用了 3 个 page 指令，设置了两个属性 language 和 import；其中第 2 个 page 指令为 import 属性指定了两个属性值，它们之间用逗号隔开。一个 JSP 页面可以有多个<%@ page import= " " %>，但只能为其他属性指定一个值。像该例中的<%@ page language= "java" %>，如果写两个，就是错误的。

下面分别来讲述各种属性。

（1）import 属性：import="包名或类名"，定义该 JSP 页面需要引入的包名，这样就可以使用这些包中的任何类。属性值可以是一个包，也可以是包中的具体的一个类。具有与 Java 的 import 相同的功能。例如：

```
<%@ page import= "java.awt.* " %>
<%@ page import= "java.util.Date" %>
```

Java 中所有的类都应该被放在包中，使用包可以防止名称冲突。默认 JSP 引入如下包：java.lang.*，javax.servlet.*，javax.servlet.jsp.*，javax.servlet.http.*等。

（2）isThreadSafe 属性：isThreadSafe=true|false，设置 JSP 页面是否可以多线程访问。其属性值只有两个，即 ture 或 false。默认为 true，JSP 页面可以同时响应多个用户的请求，这时由于线程不安全就有可能导致错误。例如，如果将 isThreadSafe 属性设置成 true，一个用户请求删除了数据库中的某条记录，其他用户同时请求读取被删除的记录，就会导致错误；如果将 isThreadSafe 属性设置成 false，系统将不会被同步访问，JSP 页面同一时刻只能响应同一用户的请求，其他用户需要排队等待，上述的错误就不会发生。

（3）session 属性：session=true|false，设置是否使用内置的 session 对象。其属性值只

有两个，即 true 或 false。默认为 true。关于 session 对象，将在 7.3 节介绍。

（4）buffer 属性：指定输出流对象 out 使用的缓存区的大小。其格式如下。

```
<%@ page buffer= "大小" %> //指定一个值，如 12KB，4KB，默认为 8KB
```

或

```
<%@ page buffer= "none" %> //没有缓存区
```

缓存区与另一属性 autoFlush 有关。

（5）autoFlush 属性：autoFluse=true|false，是配合 buffer 属性使用的。当为 true 时，表示缓存区在被装满时应该清除；当为 false 时，表示缓存区在被装满时会出现缓存溢出异常。默认为 true。

```
<%@ page autoFlush= "true" %>表示缓存区装满时应该清除。
<%@ page autoFlush= "false" %>表示缓存区装满时会出现缓存溢出异常。
```

注意：当 buffer 被设置为 none 时，autoFlush 属性值不能为 false。

（6）extends 属性：extends="类名"，声明父类，所生成的 Servlet 类将继承这个类。

（7）info 属性：定义一个字符串，该字符串可以通过 getServletInfo()方法获取。其格式如下。

```
<%@ page info="字符串" %>
```

（8）isErrorPage 属性：isErrorPage=true|false，定义当前页面是否可以作为其他 JSP 页面的错误页面，也就是该页面是否为异常处理页面。默认为 false。

（9）errorPage 属性：指如果抛出一个异常，而异常没有被捕获时，则转到 errorPage 属性所指向的 URL，通常为 JSP 页面。其格式如下。

```
<%@ page errorPage= "处理异常页面的 URL" %>
```

（10）contentType 属性：定义响应的 JSP 页面的文件格式（MIME）和编码方式。文件格式可被设置为 text/html，text/plain，application/x-msexcel，application/msword 等，如果用户想知道浏览器能支持哪些 MIME 类型，可以单击资源管理器→工具→文件夹选项→文件类型查看。MIME 类型默认为 text/html，字符编码方式 charset 默认为 ISO-8859-1。

设置 contentType 属性共有两种方式。一种是只设置文件格式，例如：

```
<%@ page contentType= "application/x-msexcel " %>//这个页面是电子表格 Excel 页面
```

一种是除了设置 contentType 属性，还设置字符编码方式，例如：

```
<%@ page contentType= "text/html ;charset=GB2312" %> //这个页面是 HTML 页面，字符编码方式为 GB2312。
```

（11）pageEncoding 属性：仅在 JSP1.2 中有效，它定义了页的字符编码方式。除非指定 page 指令中的 contentType 属性，否则默认为 ISO-8859-1。

（12）language 属性：用来设置将要使用的程序设计语言的类型，属性值只有一个，只能取 java。例如：

```
<%@ page language= "java" %>
```

例 7_5：　在 Word 文档中显示 info 属性的信息。

Ex7_5.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%@ page buffer="12kb" autoFlush="true"%>
<%@ page contentType="application/msword;charset=GB2312"%>
<%@ page info="welcome to JSP world！"%>
<html>
  <head>
    <title>My JSP 'Ex7_5.jsp' starting page</title>
  </head>
  <body>
    <%
        String str=getServletInfo();
        byte b[]=str.getBytes("ISO-8859-1");
        str=new String(b);
        out.println(str);
    %>
  </body>
</html>
```

程序的运行结果如图 7.5 所示。

图 7.5　在 Word 文档中显示 info 属性的信息

例 7_6：错误页面的用法。

Ex7_6_error.jsp

```
<html>
  <head>
    <title>错误页面</title>
  </head>
  <body>
    This following error has occurred:<%=exception.toString()%>
  </body>
```

```
</html>
Ex7_6_makeError.jsp：
<%@ page contentType="text/html;charset=gb2312" errorPage="Ex7_6_error.jsp"%>
<html>
  <head>
    <title>使用错误页面</title>
  </head>
  <body>
    <%
      String isMakeError=request.getParameter("makeError");
      if(isMakeError!=null&&isMakeError.equals("true"))
        throw new Exception("此处发生了一个错误！");
    %>
    <FORM name="form1" action="Ex7_6_makeError.jsp" method="POST">
      <INPUT type="hidden" name="makeError" value="false">
      <INPUT type="button" value="生成错误" onclick="this.form.makeError.value=true;
this.form.submit();">
    </FORM>
    <p>本程序演示在 JSP 页面的 page 指令中设置 errorPage 属性的功能。</p>
    单击下面按钮将会使本页抛出一个异常。
  </body>
</html>
```

程序的运行结果如图 7.6 所示。

图 7.6　错误页面的用法

2．include指令

include 指令是静态包含，用于在 JSP 页面中包含其他文件，程序在编译时会进行代码的替换，相当于在该 JSP 页面中的某处插入一段代码。被包含的文件可以是其他的 JSP 页面。通常 include 指令被用于在多个页面中使用同一段代码，这时可以把这段相同的代码放到一个 JSP 页面中，采用 include 指令把这个 JSP 页面包含进来。

include 指令要和后面讲述的 include 动作相区别。include 指令是一种静态包含，也就是当前的 JSP 页面 A 和被包含的 JSP 页面 B 会合成为一个新的 JSP 页面 A，所以必须保证新合成的 JSP 页面 A 符合 JSP 的语法规则。

inlcude 指令只有一个 file 属性，属性值为被包含文件的带扩展名的文件名，其格式如下。

```
<%@ include file= "文件名" %>
```

需要注意的是，被包含的文件必须和当前 JSP 页面在同一个 Web 目录下。

例 7_7：include 指令的用法。

Ex7_7_include1.jsp

```
<%@ page language="java" contentType="text/html;charset=gb2312"%>
<html>
  <head>
    <title>My JSP 'Ex7_7_include1.jsp' starting page</title>
  </head>
  <body>
    <%! String str="  这是主页面!请输入您的名字："; %>
    <%= str %>
    <%@ include file="Ex7_7_include2.jsp" %>
    <%
        String str2=request.getParameter("sname");
        response.setContentType("text/html;charset=gb2312");
        out.println("在主页面中，您刚才输入的姓名为："+str2);
    %>
  </body>
</html>
```

Ex7_7_include2.jsp

```
这是被包含页面!
<form method="POST" action="Ex7_7_include1.jsp">
<INPUT type="text" name="sname" maxlength="10">
<INPUT type="Submit" name="submit" value="提交">
</form>
```

程序的运行结果如图 7.7 所示。需要注意的是，如果在 Ex7_7_include2.jsp 文件程序的开始部分加上语句<%@ page language="java" contentType="text/html;charset=gb2312"%>，程序就会出现语法错误。因为合成后会出现两条完全一样的 page 指令，这在 JSP 中是不允许的。

图 7.7　include 指令的用法

3．taglib指令

taglib 指令声明 JSP 文件使用定制标签，指定了定义这些标签的标签库的名称，并指定了它们的标签前缀。

taglib 指令有两个属性：uri 属性是统一资源标识符，指定了标签库的位置；prefix 属性指定了前缀，对于标签库来说是唯一的前缀。

关于定制标签库的具体内容本书不进行陈述，如有需要，请参看相关书籍。

7.2.3　JSP动作

JSP 动作可以使 JSP 和其他服务器组件进行交互，影响 JSP 运行时的功能。JSP 动作的功能包括动态插入文件、动态加载 Servlet 或其他 JSP 页面、跳转到其他页面、使用 JavaBean 处理数据库访问等。一共有 7 个 JSP 动作：<jsp:useBean />、<jsp:forward />、<jsp:include />、<jsp:setProperty />、<jsp:getProperty />、<jsp:param />和<jsp:plugin />。其中，<jsp:useBean />、<jsp:setProperty />和<jsp:getProperty />结合使用，被用来在 JSP 中处理 JavaBean，分别完成创建 JavaBean、对 JavaBean 设置属性值和获取属性值，这部分内容将在第 8 章中详细介绍。下面将详细讲述剩余的 4 个动作。

1．include动作<jsp:include />

include 动作是动态包含一个文件，它是在 JSP 运行时插入一个文件。这与前面所讲的 include 指令不同。include 指令是静态包含，在编译时会合二为一形成一个新的 JSP 页面。而 include 动作是一种动态包含，JSP 页面和它所包含的文件在逻辑和语法上是完全独立的，也就是说，当 JSP 页面编译形成 Java 文件时，不会合成为一个文件，仅仅是包含了另一个文件。这相当于 C 语言中的函数调用，JSP 文件在编译时，如果遇上 include 动作，就转去执行被包含的文件，当执行完被包含的文件后，再回来继续向下执行。如果被包含的文件是文本文件，则把内容发送到客户端进行显示；如果被包含的文件是 JSP 文件，则执行这个文件，然后将执行结果发送到客户端显示。

include 动作的一般格式如下。

```
<jsp:include page="被包含文件的名字" />
```

或

```
<jsp:include page="被包含文件的名字" >
</jsp:include>
```

page 属性的属性值是被包含文件的名字（含相对路径），指向被包含文件，或者计算结果是一个关于相对路径的字符串。在第 2 种格式中，<jsp:include page="被包含文件的名字" >和</jsp:include>必须成对出现，</jsp:include>标签说明 include 动作的结束。需要注意的是，“jsp”和冒号“：”及“include”三者之间不允许有空格。

例 7_8：include 动作的用法。

Ex7_8.html

```
<html>
  <head>
    <title>Ex7_8.html</title>
  </head>
  <body bgcolor="cyan">
     请选择一个被包含的文件：
     <form name="frm" action="Ex7_8_include1.jsp" method="POST">
        <P>
       <INPUT type="radio" name="file1" value="Ex7_2">Ex7_2
       <INPUT type="radio" name="file1" value="Ex7_3">Ex7_3
       <INPUT type="radio" name="file1" value="Ex7_4">Ex7_4
       <INPUT type="radio" name="file1" value="Ex7_5">Ex7_5
       </P>
       <INPUT type="submit" name="submit" value="确定">
       <INPUT type="reset" name="reset" value="取消">
     </form><br>
  </body>
</html>
```

Ex7_8_include1.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%@ page contentType="text/html;charset=GB2312"%>
<html>
  <head>
    <title>include 动作 </title>
  </head>
  <body>
          <%
              String str=request.getParameter("file1");
              response.setContentType("text/html;charset=gb2312");
              String filesname="";
              if(str.equals("Ex7_2"))
                  filesname="Ex7_2.html";
              if(str.equals("Ex7_3"))
                  filesname="Ex7_3.jsp";
              if(str.equals("Ex7_4"))
                  filesname="Ex7_4.html";
              if(str.equals("Ex7_5"))
                 filesname="Ex7_5.jsp";
            %>
            <p>运行程序<%=filesname%>：</p>
            <jsp:include page="<%=filesname%>" />
  </body>
</html>
```

该程序首先运行 HTML 程序，在表单中，采用单选按钮提交需要包含的文件名。在 Ex7_8_include1.jsp 文件中首先得到提交的参数值，然后用 if 语句判断参数值是否等于相应的文件名，最后采用 include 动作包含用户所选用的文件名。在程序中，page 的属性值是一个表达式的值，如<jsp:include page="<%=filesname%>" />。

程序的运行结果如图 7.8 所示。

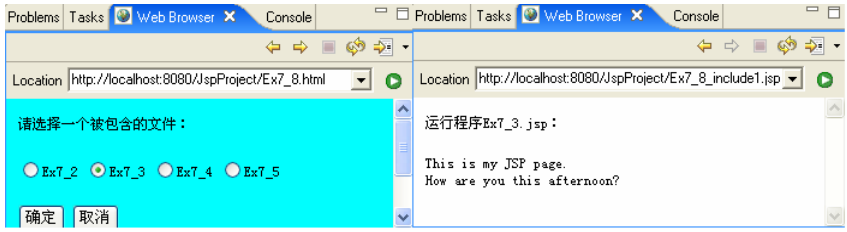


图 7.8　include 动作的用法

2．param动作<jsp:param />

param 动作一般和 include 动作、plugin 动作或 forward 动作一起使用，给这些标签提供参数。参数的形式为“参数名—参数值”。其一般格式如下。

```
<jsp:param name= "参数的名字" value= "参数的值" >
```

当将该动作和 include 动作同时使用时，可以将 param 参数传递给被包含文件。这相对于 C 语言中函数调用时，带有参数。

需要注意的是将 include 动作和 param 动作结合使用时，其格式只能如下。

```
<jsp:include page="被包含文件的名字" >
  <jsp:param name= "参数的名字" value= "参数的值" >
</jsp:include>
```

注意：此时 include 动作的格式只能是这种。因为 param 参数被传递给被包含文件，所以 param 动作只能被放在 include 动作之间，而不能采用<jsp:include page="被包含文件的名字"　/>格式，因为“/”表示 include 动作的结束。

例 7_9：将 include 动作和 param 动作结合使用。

Ex7_9.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%@ page contentType="text/html;charset=GB2312"%>
<html>
  <head>
    <title>将 include 动作和 param 动作结合使用</title>
  </head>
  <body bgcolor="yellow">
    <h1>This is my JSP page. </h1><br>
    <jsp:include page="Ex7_9_param.jsp">
```

```
            <jsp:param name="num" value="3" />
        </jsp:include>
    </body>
</html>
```

Ex7_9_param.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<html>
    <head>
        <base href="<%=basePath%>">
        <title>被包含页面 </title>
    </head>
    <body>
        <%
            String str=request.getParameter("num");
            int n=Integer.parseInt(str);
            int s=1;
            for(int i=1;i<=n;i++)
                s=s*i;
        %>
        <%= n %>的阶乘为：
        <br> <%=s %>
    </body>
</html>
```

注意：执行完文件名为 Ex7_9_param.jsp 的被包含页面后，程序会回到主页面 Ex7_9.jsp。程序的运行结果如图 7.9 所示。

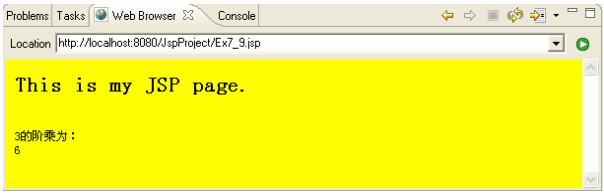


图 7.9　将 include 动作和 param 动作结合使用

3．forward动作</jsp:forward>

forward 动作实现的是页面间的跳转。如果在 JSP 中遇到 forward 动作，程序就会从该指令处停止继续执行当前页面，而转向其他的一个 JSP 页面。forward 动作和 include 动作不同，include 动作是在执行完被包含文件后，程序还会回到主页面；而 forward 动作是跳转到别处，程序不会回到主页面了。

forward 动作的格式如下。

```
<jsp:forward page="  跳转到的文件的名字  " />
```

或

```
<jsp:forward page=" 跳转到的文件的名字 ">
    <jsp:param name= "参数的名字" value= "参数的值" >
</jsp:forward>
```

param 动作为可选项，forward 动作可以带参数，也可以不带参数。需要注意的是，关键字“jsp”、冒号“:”和“forward”之间不允许有空格。

例 7_10：使用 forward 动作完成登录页面的实现。

Ex7_10_login.html

```
<html>
  <head>
    <title>Ex7_10_login.html</title>
  </head>
  <body>
  <H1 align="center"><FONT color="#ff0000">forward 动作的使用</font></H1>
    <FORM method="post" action="Ex7_10_forward1.jsp">
      <div align="center">
      <center>
      <TABLE border="0" width="50%">
          <tr>
              <TD width="50%" align="right"><font color="#f00ff0">用户名：</font></TD>
              <TD width="50%"><input type="text" name="inputName" size="20"></TD>
</tr>
          <tr>
              <TD width="50%" align="right"><font color="#f00ff0">密 码：</font></TD>
              <TD width="50%"><input type="password" name="inputPwd" size="20"></TD>
</tr>
          <TR>
              <TD width="100%" colspan="2">
               <P align="center">
<INPUT type="radio" value="manage" checked name="select">
<font collor="#ff00ff">管理</font>   
                    <INPUT type="radio" value="protect" name="select"> <font collor="#ff00ff">
维护</font>   
                    </TD></TR></TABLE></CENTER></div>
          <p align="center">
            <input type="submit" value="确定" name="submit">
            <input type="reset" value="取消" name="reset"> </FORM><HR>
</body>
</html>
```

Ex7_10_forward1.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%@ page contentType="text/html;charset=GB2312"%>
<html>
  <head>
    <title>跳转主页面 </title>
  </head>
  <body>
    <h1><FONT color=red>forward 的使用</FONT></h1><hr><hr>
    <%
        String fileName;
         String name=request.getParameter("inputName");
         String password=request.getParameter("inputPwd");
         String select=request.getParameter("select");
         if(select.equals("manage"))
            fileName="Ex7_10_forward2.jsp";
         else
            fileName="Ex7_10_forward3.jsp";
    %>
    <jsp:forward page="<%=fileName%>">
        <jsp:param name="username" value="<%= name %>"/>
        <jsp:param name="password" value="<%=password %>"/>
    </jsp:forward>
  </body>
</html>
```

Ex7_10_forward2.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="gb2312"%>
<%@ page contentType="text/html;charset=GB2312"%>
<html>
  <head>
    <title>管理页面</title>
  </head>
  <body>
<H1 align="center">
          <FONT color="#0000ff"> 这是管理页面 </FONT></H1>
<%
        String sname=request.getParameter("username");
        byte b[]=sname.getBytes("ISO-8859-1");
        sname=new String(b);
        String spwd=request.getParameter("password");
        byte c[]=spwd.getBytes("ISO-8859-1");
        spwd=new String(c);
        response.setContentType("text/html;charset=GB2312");
    %>
```

```
            <P align="center">   <FONT color="#8000ff"> 用户名：<%=sname%></FONT> </P>
        <P align="center"><FONT color="#8000ff">密  码：<%=spwd%></FONT>
            </P>
        <p  align="center"><INPUT  type="button"  name="btn"  value=" 返 回 "  onclick="history.back()"
checked="checked" >
        </body>
        </html>
```

Ex7_10_forward3.jsp

```
        <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
        <html>
          <head>
            <title>维护页面</title>
          </head>
          <body>
            <H1 align="center">
                        <FONT color="#0000ff"> 这是维护页面 </FONT>
             </H1>
             <%
                String sname=request.getParameter("username");
                byte b[]=sname.getBytes("ISO-8859-1");
                sname=new String(b);
                String spwd=request.getParameter("password");
                byte c[]=spwd.getBytes("ISO-8859-1");
                spwd=new String(c);
                response.setContentType("text/html;charset=GB2312");
             %>
             <P align="center"><FONT color="#800000">用户名：<%=sname%></FONT>
             </P>
             <P align="center"><FONT color="#800000">  密  码：<%=spwd%></FONT>
             </P>
            <p  align="center"><INPUT  type="button"  name="btn"  value="返回"  onclick="history.back()"
checked="checked" >
          </body>
        </html>
```

程序的运行结果如图 7.10 所示。

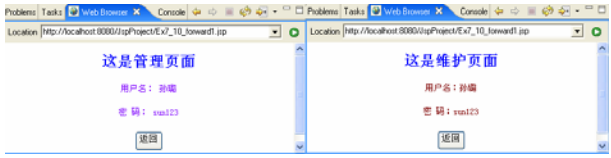


图 7.10　使用 forward 动作完成登录页面的实现

Ex7_10_login.html 页面用于显示登录页面。Ex7_10_forward1.jsp 页面用于根据条件转向其他相应的页面，即管理页面或维护页面，跳转时带两个参数 username 和 password。参数值必须是表达式的值，如 value="<%=name %>"，不能是 value="name"，因为后者传递的是字符串 name。Ex7_10_forward2.jsp 页面是管理页面，显示传递的参数，即用户名和密码。Ex7_10_forward3.jsp 页面是维护页面，其功能和管理页面相同。

4．plugin动作</jsp:plugin/>

plugin 动作用于在 JSP 页面中插入 Java Applet 小应用程序或 JavaBean。在 HTML 中，使用<applet code= "" higth= "" width= ""></applet>可以运行 Java Applet 小应用程序，但并不是所有的浏览器都支持 Applet，而 plugin 动作可以保证客户端能够运行 Applet 小应用程序。

plugin 动作的格式如下。

```
<jsp:plugin type="bean 或 applet" code=".class 文件名" codebase=".class 文件路径名" />
```

或

```
<jsp:plugin type="bean 或 applet" code=".class 文件名" codebase=".class 文件路径名" >
</jsp:plugin>
```

7.3　JSP的内置对象

内置对象是 JSP 中可以直接使用的预定义变量。不需要声明或载入。JSP 一共含有 9 个这样预定义的内置对象，分别是 request，response，pageContext，session，application，config，out，page，exception。

7.3.1　request对象

request 对象是类 java.servlet.HttpServletRequest 的一个对象，它是与请求相关的一个参数，JSP 将请求信息的内容放在 request 对象中。请求信息的内容包括利用 HTML 提交的用户信息、请求的参数名和参数值等。这些信息都可以通过 request 对象得到。

和 request 对象有关的方法大多和参数有关，参数包括表单中传递的参数、地址栏传递的参数和 JSP 的 param 动作传递的参数。

1．getParameter()

getParameter()方法是得到参数的值。该方法适用于只有一个值的参数。其格式如下。

```
字符串变量名=request.getParameter( "参数" );
```

该方法的返回值为一个字符串类型。

2．getParameterNames()

getParameterNames()方法是得到所有参数的名称，其格式如下。

```
Enumeration e=request.getParameterNames();
```

该方法的返回值为一 Enumeration 类型。

3．getParameterValues()

getParameterValues()方法是得到参数的所有值，适用于一个参数有多个值的情况，如复选框和允许多选的列表框。其格式如下。

```
String [ ]str=request.getParameterValues( "参数" );
```

该方法的返回值为一个字符串的数组。

例 7_11：getParameterValues()方法的应用。

Ex7_11.html

```
<html>
  <head>
    <title>getParameterValues()方法的应用</title>
  </head>
  <body>
   <form action="Ex7_11.jsp" method="POST">
    你的姓名：<input type="text" name="txtName">
    你所喜欢的运动：<br>
    <input type="checkbox" name="sport" value="run" checked="checked">跑步<br>
    <input type="checkbox" name="sport" value="walk"> 散步<br>
    <input type="checkbox" name="sport" value="table">乒乓球 <br>
    <input type="checkbox" name="sport" value="basketball">篮球 <br>
    <input type="checkbox" name="sport" value="football">足球 <br>
    <input type="submit" name="submit" value="提交" >
    <input type="reset" name="reset" value="取消" >
  </form>
  </body>
</html>
```

Ex7_11.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ page contentType="text/html;charset=UTF-8"%>
<html>
  <head>
    <base href="<%=basePath%>">
          <title>通过 getParameterValues()方法获取复选框的值</title>
  </head>
  <body>
```

```
<%
    response.setContentType("text/html;charset=UTF-8");
    String sports[]=request.getParameterValues("sport");
    String name=request.getParameter("txtName");
    byte b[]=name.getBytes("ISO-8859-1");
    name=new String(b);
    out.println("你的姓名是："+name+"<br>");
    out.println("你喜欢的运动是："+"<br>");
    if(sports==null)
      out.println("无");
    else
      for(int i=0;i<sports.length;i++)
      {
        if(sports[i].equals("run"))
          out.println("跑步"+"<br>");
        if(sports[i].equals("walk"))
          out.println("散步"+"<br>");
        if(sports[i].equals("table"))
          out.println("乒乓球"+"<br>");
        if(sports[i].equalsIgnoreCase("basketball"))
          out.println("篮球"+"<br>");
        if(sports[i].equalsIgnoreCase("football"))
          out.println("足球"+"<br>");
      }
  %>
 </body>
</html>
```

在 Ex7_11.html 中定义了文本框和复选框，用户可以进行多项选择，这样就构成了同一个属性名 sport 多个值的情况。在 Ex7_11.jsp 中，利用 getParameterValues()方法得到用户选择的多个值。需要注意的是，值是表单中 value 属性的值，而不是在表单中显示的“跑步”、“散步”、“乒乓球”、“篮球”、“足球”。程序的运行结果如图 7.11 所示。

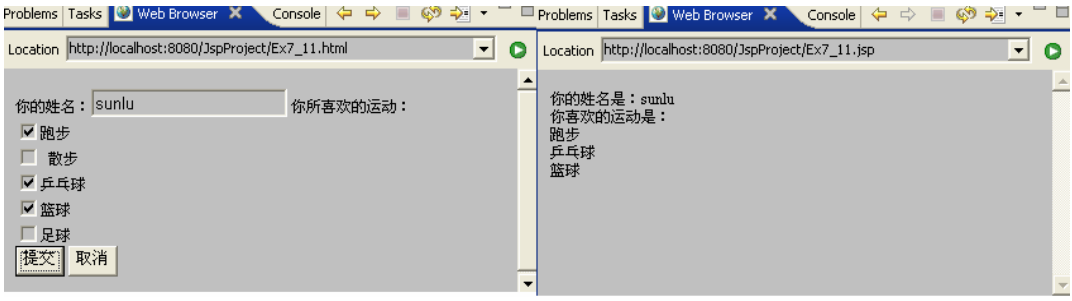


图 7.11　getParameterValues()方法的应用

4．setAttribute()和getAttribute()

这两种方法用于在多个页面中共享数据。其格式如下。

```
request.setAttribute( "参数名", "参数值" );
String str=request.getAttribute( "参数名" );
```

例 7_12：在一个页面中设置值，在另一个页面中显示。

Ex7_12_1.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>
    <title>主页面用 getAttribute()方法显示值</title>
  </head>
  <body>
    <jsp:include page="Ex7_12_2.jsp" />
      您在另一个页面中设置的值为：
      <%= request.getAttribute("gr") %>
  </body>
</html>
```

Ex7_12_2.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>
    <title>在该页面中设置值</title>
  </head>
  <body>
    <%
        request.setAttribute("gr","Hello! Everyone! ");
    %>
  </body>
</html>
```

5．getAttributeNames()

如果用多个 setAttribute()方法给多个参数设置了值，则可以用 getAttributeNames()方法获得多个参数。其格式如下。

```
Enumeration e=request.getAttributeNames();
```

例 7_13：在一个页面中设置多个值，在另一个页面中显示。

Ex7_13_1.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
```

```
    <head>
      <title>使用 getAttributeNames()方法得到在另一个页面中设置的多个值</title>
    </head>
    <body>
        <jsp:include page="Ex7_13_2.jsp" />
        <%
            Enumeration e=request.getAttributeNames();
            while(e.hasMoreElements())
            {
                out.println();
                String name=(String)e.nextElement();
                String value=(String)request.getAttribute(name);
                out.print("变量名称："+name);
                out.println("变量内容："+value);
            }
        %>
    </body>
</html>
```

Ex7_13_2.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>
    <title>在该页面中设置多个值 </title>
  </head>
  <body>
    <%
        request.setAttribute("a1","111");
        request.setAttribute("a2","222");
        request.setAttribute("a3","333");
    %>
  </body>
</html>
```

6．获得客户端和服务器端的各种信息

request.getRemoteAddr() 获得客户端的 IP 地址。

request.getRemoteHost() 获得客户机的名称，如果得不到，就获得 IP 地址。

request.getServerName () 获得服务器的名称。

request.getServerPort () 获得服务器的端口号。

例 7_14：获得客户端和服务器端信息。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
```

```
    <head>
      <title>My JSP 'Ex7_14.jsp' starting page</title>
    </head>
    <body>
      This is my JSP page. <br>
      <b>你的机器的 IP 地址为：<%= request.getRemoteAddr() %></b><br>
      <B>你的机器的名称为：<%= request.getRemoteHost() %></B><br>
      <B>服务器的名称为：<%= request.getServerName() %></B><br>
      <B>服务器的端口号为：<%= request.getServerPort()%></B>
    </body>
</html>
```

程序的运行结果如图 7.12 所示。

图 7.12　获得客户端和服务器端信息

7.3.2　response对象

response 对象是类 javax.http.HttpServletResponse 的一个对象。它有几个常用方法。

1．动态设置响应的MIME类型

当用户请求访问一个 JSP 页面时，可以用 page 指令设置页面的 contentType 属性，它是静态的。若想动态改变 contentType 属性值，需用如下方法。

```
response.setContentType("String s") ;
```

该方法动态设置响应的 MIME 类型，参数 s 可以取 text/html，text/plain，application/x-msexcel，application/msword 等。

2．设置响应头

HTTP 协议是基于请求响应模型的，首先客户端与服务器建立连接，然后发送 HTTP 请求消息，服务器会处理请求，并将响应消息传送给客户端。请求消息和响应消息都包括请求行、HTTP 头和消息体。以下是一个 HTTP 请求消息的例子。

```
POST/Certify HTTP/1.1
Host 129.168.1.1
Content-Type:text/html
```

```
Content-Length:16

Hello! Everyone!
```

在这个例子中，前两行是请求行，包括 post 方法、HTTP 版本号、主机地址；中间两行是 HTTP 头，包括消息内容的类型和长度的定义；空行后是消息体，表示传输消息的内容。请求消息和响应消息的格式和组成非常相似。用 response 对象可以动态添加响应头。其方法如下。

```
response.setHeader( String head,String value ) ;
```

或

```
response.addHeader(String head,String value ) ;
```

例 7_15：添加响应头完成网页自动刷新。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>
    <title>response  刷新页面</title>
  </head>
  <body>
     <%
         response.setHeader("refresh","3");
         out.println(new Date().toLocaleString());
     %>
  </body>
</html>
```

3．重定向

response 对象的最常用方法是将用户的请求转去其他页面。例如，用户在表单中输入的信息不正确，提交给服务器后，会要求重新输入信息，这时就需要回到输入信息页面，会用到如下方法。

```
response.sendRedirect(网页地址) ;
```

例 7_16：在表单中输入姓名和年龄，如果没有输入或对年龄输入的不是数字，则需要重新输入。

Ex7_16.jsp，定义表单。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>
    <title>My JSP 'Ex7_16.jsp' starting page</title>
```

```
  </head>
  <body>
    <form action="Ex7_16_response.jsp" method="POST">
      姓名：<INPUT type="text" name="name" >
      年龄：<INPUT type="text" name="age" value="0" maxlength="10">
      <input type="submit" name="submit" value="提交">
      <INPUT type="reset" name="reset" value="重置">
    </form>
  </body>
</html>
```

Ex7_16_response.jsp，对用户输入的信息进行处理。

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>

    <title>My JSP 'Ex7_16_response.jsp' starting page</title>
  </head>
  <body>
    <%
        int flag=0;
        response.setContentType("text/html;charset=UTF-8");
        String sname=request.getParameter("name");
        String sage=request.getParameter("age");
        if(sage.length()>=3)
          flag=1;
        for(int i=0;i<=sage.length()-1;i++)
        {
          char s=sage.charAt(i);
          if(s<'0'||s>'9')
          {
            flag=1;
            break;
          }
        }
        if(sname==null||sage==null)
          response.sendRedirect("Ex7_16.jsp");
        else if(flag==1)
          response.sendRedirect("Ex7_16.jsp");
        else
          out.println("输入信息正确！");
    %>
    <form >
```

```
            <INPUT type="button" name="btnback" value="返回" onclick=history.back()>
        </form>
    </body>
</html>
```

7.3.3　session对象

1．session的概念

session 是类 javax.servlet.http.HttpSession 的一个对象。session 是指客户端与服务器之间的一次会话。会话从连接到服务器开始，到与服务器断开连接为止，在这期间都只给该客户端分配一个 session。session 是以“名称数值”对的形式保存的。当客户端第一次请求 JSP 页面时，服务器就会发送一个会话标识符 ID 给客户端，此时浏览器使用 Cookie 保存这个会话标识符。该会话标识符是一个唯一的用户标识符，此时，session 对象就会常驻内存，等待同一用户通过这个会话标识符再一次调用同一个 session。

当两个客户端 A 和 B 访问同一个 JSP 页面时，创建的是两个不同的 session；如果是一个客户端访问同一个 JSP 页面，打开了两个浏览器窗口，创建的是两个线程，但是同一个 session；如果是一个客户端访问两个不同的 JSP 页面，虽然访问的 JSP 不同，但因为只有一个客户端，服务器为这个客户端创建一个 session 对象。

2．session超时

HTTP 协议是无状态的协议，当用户不再访问应用程序时，并不会发给服务器终止的信号。所以，服务器并不知道用户是否还要访问应用程序，如果不采取某种方法解决这个问题，内存中就会积累大量的 session 对象。为此，Servlet 采用“超时限制”的办法来判断用户是否还在访问：如果某个用户在一定时间之内没有发出后继请求，则该用户的会话被作废，那么其 session 对象就会被释放。会话默认的“超时时间”由 Servlet 容器定义。

超时时间这个值可以通过 getMaxInactiveInterval()方法获得，通过 setMaxInactiveInterval()方法可以修改这个值，这些方法中的超时时间以秒计算。如果会话的超时时间值被设置为－1，则会话永不超时。

可以使用 invalidate()方法释放掉这个用户的所有 session 数据，也可以自动作废。作废会话意味着从内存中删除 session 对象及它的数据。例如，如果在一定时间之内（默认为 30 分钟）用户不再发送请求，Web 应用服务器会自动作废这个用户的会话。

3．和session有关的常用方法

isNew()判断是否是新创建的 session。如果是新创建的，返回 true，否则返回 false。

getValue(String name)　得到参数的值。

putValue(String name，String Value)　设置参数和参数的值。

removeValue(String name)　删除指定的参数。

getValueNames()得到所有参数。

getCreationTime()获得创建 session 的时间，单位是毫秒。

getID()获得该 session 的 ID。

getLastAccessedTime()获得当前请求的最后一次访问时间，

setAttribute(String name, String value)设置参数的值，将一对 name/value 属性保存到 session 对象中。

getAttribute(String name)得到参数的值。

例 7_17：session 的用法。

Ex7_17 .jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>
    <title>session 主页面</title>
  </head>
  <body>
    <%
        String usename="tom";
        String password="123456";
        session.putValue("usename",usename);
        session.putValue("password",password);
    %>
    <A href="Ex7_17_1.jsp">指向第 2 页</A>
  </body>
</html>
```

Ex7_17_1.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head>
    <title>session 转向页面 1</title>
  </head>
  <body>
    <%
        String usr=(String)session.getValue("usename");
        String pwd=(String)session.getValue("password");
    %>
    usename 参数的值为：<%=usr %><br>
    password 参数的值为：<%=pwd %> <br>
    <%
        out.println("session 创建时间:"+session.getCreationTime()+"<br>");
        out.println("session 用户标识符 id:"+session.getId()+"<br>");
        out.println("session 最后访问时间:"+session.getLastAccessedTime()+"<br>");
        out.println("session 原来最大休眠时间："+session.getMaxInactiveInterval()+"<br>");
        int time=session.getMaxInactiveInterval()+1;
        session.setMaxInactiveInterval(time);
```

```
                out.println("设置最大休眠时间："+session.getMaxInactiveInterval()+"<br>");
                String []name=session.getValueNames();
                for(int i=0;i<name.length;i++)
                {
                   out.println(session.getValue(name[i])+"<br>");
                   out.println("<br>");
                }
                session.removeValue("usename");
            %>
            <a href="<%=response.encodeURL("Ex7_17_2.jsp")%>">转向第 3 个页面
            </a>
        </body>
    </html>
```

Ex7_17_2.jsp

```
    <%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
    <html>
        <head>
            <title>session 转向页面 2</title>
        </head>
        <body>
            <%
                String usr=(String)session.getValue("usename");
                String pwd=(String)session.getValue("password");
            %>
            usename 参数的值为：<%=usr%><br>
            password 参数的值值为：<%=pwd%>
        </body>
    </html>
```

7.3.4　application对象

session 对象给不同的客户端分配的是不同的 session 对象。和 session 对象不同的是，application 对象给不同的客户端分配的是相同的 application 对象。

7.3.5　page对象

page 对象代表当前页面，在 JSP 中使用时与 Java 的 this 对象等价，用于引用当前对象。

7.3.6　out对象

out 对象用于输出，用于向用户输出数据。在前面的例子中已经多次使用过 out 对象。out 对象提供了 print()方法和 println()方法。大多数时候不需要调用 out 对象的方法来输出，而是把要输出的内容直接添加到 JSP 中；当在一大块 Java 代码中需用输出语句时，最好还是用 out.print()或 out.println()，这样程序代码紧凑，可读性强。

7.3.7　pageContext对象

pageContext 是 java.servlet.jsp.PageContext 类的对象。这个类允许访问页面属性，并将请求对象传递到应用程序组件。

7.3.8　exception对象

通过 exception 对象向错误处理页面传递异常。

7.4　JSP在项目中的应用

前面讲解了 JSP 的概念及用法。现在以人力资源项目中系统管理模块的用户组管理实现为例讲解 JSP 在项目中的具体应用。

在第 5 章中讲解了用户组管理页面 Servlet 的用法，现在对用户组管理页面 groupManage.jsp 进行详细讲解。用户组管理页面如图 7.13 所示。

图 7.13　用户组管理页面

groupManage.jsp 的主要源代码如下。

```
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312" import="java.sql.*"%>
<%@ page import="javax.naming.*,com.etop.bank.comm.dbcom.*"%>
<%@ page import="com.etop.bank.groupmanage.bean.*"%>
<%@ page import="com.etop.bank.groupmanage.dao.*"%>
<%@ page import="com.etop.bank.groupmanage.business.*"%>
<HTML>

    <BODY>
        <!--action 的值为处理查询用户组的 Servlet 的 URL-->
        <FORM action="../../bank/GroupManageAction" method="post">
            当前位置：用户组管理
```

```
<BR>
用户组名：
<INPUT type="text" name="group_name" size="20">
用户组码：
<INPUT type="text" name="group_id" size="20">
<INPUT type="submit" name="sub" value="查询">
<INPUT type="Submit" name="sub" value="查看全部">
<BR>
<!--location 后的值为处理该操作的 JSP 页面或 Servlet 的 URL-->
<INPUT type="button" name="sub" value="添加新用户组"
    onclick="javascript:window.location='../../bank/UserGroup/addGroup.jsp'">
<INPUT type="submit" name="sub" value="删除用户组">
<BR>
<TABLE border="1" width="892" height="345">
    <TBODY>
        <%
            ResultSet rs = (ResultSet) session.getValue("session");
            GroupBean groupBean = new GroupBean();
            GroupManageDAO gmi = GroupManageDAOIMP.getFactory().
            getImp(groupBean);
            if (rs == null) {
                rs = gmi.getAllInfo();
                if (rs == null || !rs.next())
                    out.print("数据中没有信息！ ");
                else {
        %>
        <TR>
            <TD colspan="2">
                用户组名
            </TD>
            <TD>
                用户组码
            </TD>
            <TD>
                用户组说明
            </TD>

            <TD align="center">
                操作
            </TD>
        </TR>
        <TR>
            <TD colspan="2">
                <INPUT type="checkbox" name="groupids"
```

```
                                        value="<%=rs.getString("group_id")%>">
                                    <%=rs.getString("group_name")%>
                                </TD>
                                <TD name="group_id">
                                    <%=rs.getString("group_id")%>
                                </TD>
                                <TD name="group_description">
                                    <%=rs.getString("group_description")%>
                                </TD>

                                <TD>
                                    <a

    onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
>【修改基本信息】</a><a

    onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
1'">【删除】</a>
                                </TD>
                            </TR>
                            <%
                            while (rs.next()) {
                            %>
                            <TR>
                                <TD colspan="2">
                                    <INPUT type="checkbox" name="groupids"
                                        value="<%=rs.getString("group_id")%>">
                                    <%=rs.getString("group_name")%>
                                </TD>
                                <TD name="group_id">
                                    <%=rs.getString("group_id")%>
                                </TD>
                                <TD name="group_description">
                                    <%=rs.getString("group_description")%>
                                </TD>

                                <TD>
                                    <a

    onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
>【修改基本信息】</a>
                                    <a

    onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
```

```
1'">【删除】</a>
                                        </TD>
                                    </TR>
                                    <%
                                            }
                                            }
                                        } else if (!rs.next())
                                            out.print("没找到您需要的信息！ ");
                                        else {
                                    %>
                                    <TR>
                                        <TD colspan="2">
                                            <INPUT type="checkbox" name="userGroup">
                                            用户组名
                                        </TD>
                                        <TD>
                                            用户组码
                                        </TD>
                                        <TD>
                                            用户组说明
                                        </TD>

                                        <TD align="center">
                                            操作
                                        </TD>
                                    </TR>
                                    <TR>
                                        <TD colspan="2">
                                            <INPUT type="checkbox" name="groupids"
                                                value="<%=rs.getString("group_id")%>">
                                            <%=rs.getString("group_name")%>
                                        </TD>
                                        <TD name="group_id">
                                            <%=rs.getString("group_id")%>
                                        </TD>
                                        <TD name="group_description">
                                            <%=rs.getString("group_description")%>
                                        </TD>

                                        <TD>
                                            <a

    onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
>【修改基本信息】</a><a
```

```
    onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
1'">【删除】</a>
                                        </TD>
                                    </TR>
                                    <%
                                    while (rs.next()) {
                                    %>
                                    <TR>
                                        <TD colspan="2">
                                            <INPUT type="checkbox" name="groupids"
                                                value="<%=rs.getString("group_id")%>">
                                            <%=rs.getString("group_name")%>
                                        </TD>
                                        <TD name="group_id">
                                            <%=rs.getString("group_id")%>
                                        </TD>
                                        <TD name="group_description">
                                            <%=rs.getString("group_description")%>
                                        </TD>

                                        <TD>
                                            <a

    onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
>【修改基本信息】</a>
                                            <a

    onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
1'">【删除】</a>
                                        </TD>
                                    </TR>
                                    <%
                                        }
                                        }
                                    %>
                                </TBODY>
                            </TABLE>
                        </FORM>
                    </BODY>
                </HTML>
```

现在对 groupManage.jsp 里的部分代码进行详细分析。

首先 JSP 页面是通过在传统的 HTML 网页中加入 Java 程序片断和使用各种 JSP 标签

构成的。在代码的最顶端用 JSP 指令来设置 JSP 页面的各种属性，可以设置 JSP 页面的字符编码方式，可以嵌入一个文件，可以导入标签库等。JSP 指令一共有 3 类，包括 page 指令、include 指令和 taglib 指令。在该源程序中使用 JSP 的 page 指令，如下所示。

```
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312" import="java.sql.*"%>
<%@ page import="javax.naming.*,com.etop.bank.comm.dbcom.*"%>
<%@ page import="com.etop.bank.groupmanage.bean.*"%>
<%@ page import="com.etop.bank.groupmanage.dao.*"%>
<%@ page import="com.etop.bank.groupmanage.business.*"%>
```

在以上 page 指令中指定了页面的属性，例如，语言为 Java，文件格式为 HTML 中的文本类型，字符编码方式为 GB2312。后边又利用 page 指令的 import 属性将页面里需要用到的各种包引入页面，例如，引入数据库类 javax.naming.*和 com.etop.bank.comm.dbcom.*"，引入 JavaBean 类 com.etop.bank.groupmanage.bean.*，引入接口访问对象 DAO 类 com.etop.bank.groupmanage.dao.*，引入业务实现 business 类 com.etop.bank.groupmanage. business.*。接下来在<body>...</body>内添加表单，如下所示。

```
用户组名：<INPUT type="text" name="group_name" size="20">
用户组码：<INPUT type="text" name="group_id" size="20">
                <INPUT type="submit" name="sub" value="查询">
                <INPUT type="Submit" name="sub" value="查看全部">
                <BR>
                <!--location 后的值为处理该操作的 JSP 页面或 Servlet 的 URL-->
                <INPUT type="button" name="sub" value="添加新用户组"
    onclick="javascript:window.location='../../bank/UserGroup/addGroup.jsp'">
                <INPUT type="submit" name="sub" value="删除用户组">
                <BR>
```

表示可以按照输入的用户组名或用户组码进行查询或查看全部，也可以添加新用户组和删除用户组。

在表单显示的下方需要显示查询的内容。在页面中怎么显示查询的内容？数据从何而来？于是在 JSP 中使用<%…%>标注需要执行的 Java 语句块。<%…%>可以包含任何数量的语句，可以包含变量、方法和类的定义，表达式也可以被放到其中。这样只要在<%…%>内放入 Java 代码即可获取后台数据。例如，在 JSP 中插入如下 Java 片段即可获取后台数据。

```
<%
//通过 session 获取后台数据库集合值，然后赋给结果集对象 rs
        ResultSet rs = (ResultSet) session.getValue("session");
        GroupBean groupBean = new GroupBean();
GroupManageDAO gmi = GroupManageDAOIMP.getFactory().getImp(groupBean);
                    if (rs == null) {
                        rs = gmi.getAllInfo();
                        if (rs == null || !rs.next())
```

```
                                    out.print("数据中没有信息！");
                                else {
                %>
```

接下来需要对获取的结果集进行分行显示，部分代码如下。

```
                        <%
                            while (rs.next()) {
                            %>
                            <TR>
                                <TD colspan="2">
                                    <INPUT type="checkbox" name="groupids"
                                        value="<%=rs.getString("group_id")%>">
                                    <%=rs.getString("group_name")%>
                                </TD>
                                <TD name="group_id">
                                    <%=rs.getString("group_id")%>
                                </TD>
                                <TD name="group_description">
                                    <%=rs.getString("group_description")%>
                                </TD>
                                <TD>
                                    <a

    onclick="javascript:window.location='../../bank/UserGroup/updateGroup.jsp?id=<%=rs.getString("id")%>'"
>【修改基本信息】</a>
                                    <a

    onclick="javascript:window.location='../../bank/GroupManageAction?id=<%=rs.getString("id")%>&sign=
1'">【删除】</a>
                                </TD>
                            </TR>
                            <%
                                }
                        %>
```

要逐行显示所有数据，就需要用 Java 代码 while(rs.next()),需要将此代码放入<%…%>内，而 HTML 中的数据需要被写在<%…%>外边。为了输出每一行值，需要在 JSP 中使用<%=…%>实现。这表示输出语句。例如，<%=rs.getString("group_name")%>表示输出 rs 集合里的用户组名。这样就完成了数据在 JSP 中的显示。

在 JSP 中还有很多其他属性和指令需要掌握，在这里就不一一讲述了。但需要明确一个道理：在 JSP 的早期开发中为了在页面显示数据是可以采用上面方式进行处理的，但这样做有一个不好之处就是页面代码很混乱，既有 HMTL 代码，又有 Java 代码，代码维护起来很困难。在以后的学习中将提出 MVC 概念，特别是 model2 的应用，将使得显示层与

业务逻辑层完全分开，所有数据处理在后台完成，前台 JSP 只管显示即可，不用在 JSP 中使用<%…%>引入 Java 后台数据，而是通过标签实现显示，如 jstl 标签、struts1 标签、struts2 标签。这些内容将在后续课程中学到，现在需要将 JSP 中的 JSP 指令及 JSP 脚本等内容熟练掌握，为以后学习打下基础。

7.5　实训操作

在此次实训中，主要是要熟练掌握 JSP 的基本概念、include 指令的使用、getParameter() 方法的使用、forward 动作的使用、内置对象 response 和 session 的使用等。

现在利用上边的内容完成一个添加用户信息的页面。要添加的用户信息包括姓名、性别、学历、专业、兴趣爱好、工作经历、奖惩情况等。添加成功后跳转到另外一个页面显示添加的用户信息。

习　　题

一、多项选择题

1．JSP 页面包括以下哪些元素？（　　）

A．JSP 指令

B．JSP 动作

C．JSP 脚本

D．JSP 控件

2．JSP 生命周期可以分为以下哪些阶段？（　　）

A．编译、装载

B．创建、转换

C．初始化、执行

D．销毁、卸载

3．有关 page 指令的属性理解正确的是（　　）。

A．page 指令中的 errorPage 属性是指明如果抛出一个异常，而异常没有被捕获时，此错误处理所指向的 URL。URL 可以指向一个普通的 HTML 页面。

B．page 指令的 contentType 属性用于设置 HTTP 响应头的 Content-Type。

C．pageEncoding 属性仅在 JSP1.2 中有效，它定义了页的字符编码方式。除非指定 page 指令中的 contentType 属性，否则默认为 ISO-8859-1。

D．language 属性指定将要使用的程序设计语言。

4．<%@ include file= "文件名"%>为（　　）。

A．include 指令，是静态包含。

B．include 指令，是动态包含。

C．include 动作，是静态包含。

D．include 动作，是动态包含。

5．在 JSP 页面中传递参数可以采取的方式为（　　）。

A．超链接。

B．采用 param 动作。

C．采用 setAttribute()方法。

D．以上都可以。

6．下面说法中正确的是（　　）。

A．forward 动作可以实现页面之间的跳转，执行完新页面后，会回到旧页面继续执行。

B．forward 动作可以实现页面之间的跳转，执行完新页面后，不会回到旧页面继续执行。

C．include 动作可以实现页面之间的跳转，执行完新页面后，会回到旧页面继续执行。

D．include 动作可以实现页面之间的跳转，执行完新页面后，不会回到旧页面继续执行。

7．JSP 动作有哪些？（　　）

A．forward

B．include

C．param

D．plugin

8．以下有关 JSP 脚本的理解正确的是（　　）。

A．JSP 脚本允许将 Java 代码插入到 JSP 即将生成的 Servlet 中。

B．采用<%=Expression %>形式，将表达式的值插入到 Servlet 的输出中。

C．<% code %>形式的 Java 语句块，被插入到 Servlet 的_jspService()方法中。

D．<%！code %>形式的注释，被插入到 Servlet 类中，通常用于提供注释。

9．在 JSP 中声明变量所采用的是（　　）。

A．<%…%>

B．<%!…%>

C．<%=…%>

D．<%--…--%>

二、编程题

1．创建 3 个 JSP 页面（登录、欢迎、退出）。在登录页面中，用户输入姓名和密码登录，登录成功后进入欢迎页面；在欢迎页面中显示“XXX（用户名），欢迎你”，欢迎页面包含“退出”按钮，单击“退出”按钮，跳转到退出页面；在退出页面中显示提示信息“XXX（用户名），欢迎你下次再来”，包含“退出”和“返回”按钮，单击“退出”按钮则关闭页面，单击“返回”按钮则转向登录页面。

2．创建两个 JSP 页面。在第 1 个 JSP 页面中完成注册，注册信息包括姓名、性别、年龄、出生日期、家庭住址等；在第 2 个 JSP 页面中显示刚才的注册信息。

第 8 章　JavaBean

📖 知识点

- JavaBean 的原理
- JavaBean 的创建
- 设置 JavaBean 的属性值
- 获取 JavaBean 的属性值

📣 难点

- 获取 JavaBean 的属性值
- 设置 JavaBean 的属性值

✍ 掌握

- JavaBean 的编写和使用
- 获取和设置 JavaBean 的属性值

任务引入：

在软件项目开发中，最好的设计方式是将 Web 页面设计与软件设计完全分开，即在 JSP 中尽可能少地使用 Java 代码。这样做的好处是便于系统开发与维护。那么如何将 Java 代码与页面分开处理呢？那就需要使用 JavaBean 作为中间载体，Java 代码的具体实现是在 JavaBean 中调用 set 方法完成的，JSP 只需要通过动作调用 JavaBean，然后调用 get 方法即可得到所生成的结果。

本章主要讲解 JavaBean 的基本概念、执行原理及 JavaBean 的属性等重要内容。最后再结合实际项目讲解 JavaBean 的用法。

8.1　JavaBean概述

JavaBean 是一个可重复使用的软件组件模型，是一种抽象的技术规范。JavaBean 简称为 Bean，是根据这种规范创建的具体组件。实际上，JavaBean 是遵循某种协议的 Java 类。它和 Applet 很相似，有自己的属性，方法和事件。属性可以是基本数据类型和其他的 Java 对象，其属性可以通过公有的 get 和 set 方法来定义；方法是一个动作、行为或服务；事件是 JavaBean 对自身状态改变的响应，这和 Java 中的 AWT 相似，通常是有用户发出的。

JavaBean 是基于 Java 语言的，具有以下优点。

- 由于是 Java 类，所以具有“编写一次、随处运行”的跨平台特性。
- 容易编写、维护和使用。
- 可以实现代码重用。

使用 JavaBean 的一大好处是可以把 Web 页面的设计和软件的设计分开。一个 JSP 是由 HTML 和 Java 程序段组成的，如果它们大量地交互在一起，就会显得页面混乱，不易

维护。这就需要把 Java 代码从 JSP 中分离出来，那么把它放到什么地方呢？可以将数据处理过程用一个或多个 JavaBean 来完成，在 JSP 中只需要调用该 JavaBean 就可以了。

8.2　JavaBean的编写和使用

8.2.1　JavaBean的编写

JavaBean 分为可视化组件和非可视化组件两类。在 Java 中主要使用非可视化组件。要写出一个正确的 JavaBean，必须遵循如下原则。

- JavaBean 必须是一个公共类。
- 类中必须有一个不带参数的构造方法，并且是公共方法。
- 类中含有成员变量，即属性。它们都是私有的。可获取或设置属性值，所使用方法的名称采用 get 或 set 加属性名的形式。例如，属性名为 xxx，获取属性值的方法为 getXxx()，设置属性值的方法为 setXxx()。需要注意的是，属性名一般都为小写，而在获得和设置方法的名称中，第二个单词首字母需要大写。
- 对于布尔类型的属性，可以用 isXxx()方法。
- 如果在属性名前加上关键字 Readable，表示是可读属性，不能修改；如果在属性名前加上关键字 Writeable，表示是可写属性，可以修改。

例 8_1：构造一个 JavaBean 的类。

```
package pack;
public class Rect
{
    int height,width;
    public Rect()
    {
        height=width=1;

    }
    public Rect(int h,int w)
    {
        height=h;
        width=w;
    }
    public int getHeight()
    {
        return height;
    }
    public int getWidth()
    {
```

```
        return width;
    }
    public void setHeight(int h)
    {
        height=h;
    }
    public void setWidth(int w)
    {
        width=w;
    }
    public int area()
    {
        int s=height*width;
        return s;
    }
    public int length()
    {
        int l=2*(height+width);
        return l;
    }
}
```

在这个 JavaBean 中，定义了一个矩形（Rect）类。它含有两个属性：height 和 width。成员方法 setHeight(int h)和 setWidth(int w)用来设置属性值，成员方法 getWidth()和 getHeight()用来获取属性值。

注意：一般 JavaBean 都被放在包中。该 Rect.java 文件必须经过编译，将生成的 Rect.class 文件放在新生成的 pack 文件夹内。如果想要服务器的所有 Web 目录下的 JSP 都可以使用该 JavaBean，则必须把含有 Rect.class 文件的文件夹 pack 复制到 Tomcat\classes 目录下面；一般只需要 BeanProject 下面的 JSP 使用 JavaBean，这时需要把含有 Rect.class 文件的文件夹 pack 放到 Tomcat 所在的 Tomcat5.0\webapps\ BeanProject\WEB-INF\classes\目录下。

8.2.2　JavaBean的使用

一个 JSP 文件包含两部分内容：一部分是 HTML 内容，也就是静态的 Web 内容；另一部分是动态内容，动态内容的一个重要来源就是 JavaBean。JSP 为这些动态内容提供了简单的语法，这就是 JSP 动作。

对于定义好的 JavaBean，在 JSP 中添加该 JavaBean 需要使用 JSP 动作<jsp:useBean/>，其格式如下。

```
<jsp:useBean id= "JavaBean 的对象名" class= "JavaBean 的类名" scope= "page|request|session|application " />
```

或

```
<jsp:useBean id= "JavaBean 的对象名" class= "JavaBean 的类名" scope= "page|request|session|application " >
</jsp:useBean>
```

<jsp:useBean/>动作标签实际上是创建 JavaBean 的对象。id 属性用于指定 JavaBean 的对象名，class 属性指出 JavaBean 的类名，scope 属性指出了所创建的 JavaBean 对象的使用范围，可以是 page、request、session 或 application。page 表示该 JavaBean 的有效范围是当前页面，当用户离开这个页面时，JSP 就取消该 JavaBean；request 表示该 JavaBean 的有效范围是用户的请求过程，当 JSP 作出响应后，取消该 JavaBean；session 表示该 JavaBean 的有效范围是从用户开始上网到结束的一个会话期间，也就是说，如果用户在多个页面之间相互链接，每个页面都使用了 JavaBean，并且是同一个对象名，作用范围都是 session，则该用户在这些页面中得到的是同一个 JavaBean；application 表示该 JavaBean 的有效范围是整个应用程序。所有用户共享同一个 JavaBean。当关闭服务器后，该 JavaBean 才被取消。在默认情况下，scope 取值为 page。在进行项目开发时，如果要计算网页的点击率，scope 应为 application；如果要使用 JavaBean 计算一个用户登录一次服务器浏览了多少个网页，scope 应为 session。

例 8_2：使用 JavaBean。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312"%>
<%@ page import="pack.Rect.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>使用一个 JavaBean</title>
</head>
<body>
    <jsp:useBean id="rect1" class="pack.Rect" scope="page">
    </jsp:useBean>
    <%
        rect1.setHeight(4);
        rect1.setWidth(3);
    %>
    <p> 矩形的长和宽分别是：
    <%=rect1.getHeight() %>
    <%=rect1.getWidth() %>
    <p>矩形的周长为：
    <%=rect1.length() %>
    <p>矩形的面积为：
    <%= rect1.area() %>
</body>
</html>
```

程序的运行结果如图 8.1 所示。

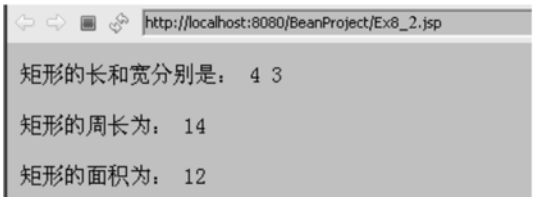


图 8.1　使用 JavaBean

该程序需要被放到 Tomcat 所在的 Tomcat5.0\webapps\ BeanProject 目录下。

在使用 JavaBean 时，必须在 JSP 文件中用<%@ page import="pack.Rect.*" %>引入 JavaBean 所在的包。并且 useBean 动作<jsp:useBean/>的 class 属性必须指出包路径。

创建 JavaBean 的过程，实际上当 JSP 执行过程中遇到 useBean 动作时，首先得到 scope 属性的值，然后在 scope 指明的范围内根据 id 指明的名称查找这个 JavaBean。如果找到了，则取回 JavaBean 对象。如果没有找到，则在 useBean 中查找 beanName 属性，如果指明了 beanName 属性，则根据该属性指明的文件去寻找永久性数据，从永久性数据中实例化指明的 JavaBean；如果没有指明 beanName 属性，JSP 会查询是否指明了 class 属性，如果指明了 class 属性，则会创建一个新的 JavaBean 实例，反之会抛出一个异常。

8.3　获取和设置JavaBean的属性值

8.3.1　获取JavaBean的属性值

获取 JavaBean 的属性值可以采用两种方式。一种方式是如例 8_1 所示的那样直接通过 JavaBean 的对象调用该 JavaBean 中的各种方法，实际上就是在 JSP 页面中嵌入 Java 代码，这样会使 JSP 页面看起来比较混乱，达不到动静分离的目的；另一种方式是采用 JSP 的动作<jsp:getProperty/>，通过该动作可以获取 JavaBean 的属性值，并把这个值用字符串的形式显示给用户。其一般格式如下。

```
<jsp:getProperty name= "JavaBean 的对象名" property= "属性名" />
```

或

```
<jsp:getProperty name= "JavaBean 的对象名" property= "属性名" >
</jsp:getProgerty>
```

name 取值是 JavaBean 的对象名，用来指定要获取哪个对象的 JavaBean 的属性值；property 具体指出该对象众多属性中的哪一个属性。该动作相当于调用 getXxx()方法。

例 8_3：获取 JavaBean 的属性值。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
```

```
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312"%>
<%@ page import="pack.Rect.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>getProperty 动作获取属性值</title>
</head>
<body>
    <jsp:useBean id="rect2" class="pack.Rect" scope="page">
    </jsp:useBean>
    <%
      rect2.setHeight(10);
      rect2.setWidth(20);
    %>
    <p> 矩形的长为：
    <jsp:getProperty name="rect2" property="height"/>

    <p>矩形的宽为：
    <jsp:getProperty name="rect2" property="width"/>
    <p>矩形的周长为：<%=rect2.length() %>
    <p>矩形的面积为：<%=rect2.area() %>
</body>
</html>
```

程序的运行结果如图 8.2 所示。

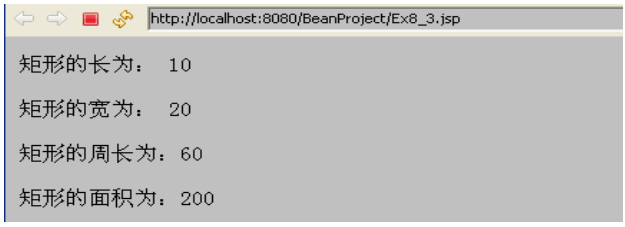


图 8.2　采用 JSP 动作获取 JavaBean 的属性值

在该程序中，获取 JavaBean 的属性 height 和 width 的值采用 JSP 动作<jsp:getProperty/>。需要注意的是，在使用该 JSP 动作前，必须使用 JSP 动作<jsp:useBean/>以获得一个 JavaBean。在<jsp:useBean/>动作中，JavaBean 的对象由 id 属性指出的，而在<jsp:getProperty/>动作中，JavaBean 的对象是由 name 属性指定的，并且 name 属性的值需要和 id 属性的值一致。实际上，<jsp:useBean/>动作相当于定义 JavaBean 的对象 rect2，而<jsp:getProperty/>动作相当于使用该对象，调用该对象的各种方法，例如，调用该 rect2 对象的 rect2.getHeight()和 rect2.getWidth()方法。程序中的 length()和 area()方法不能采用 JSP 动作<jsp:getProperty/>调用，只能采用 Java 代码 rect2.length()和 rect2.area()方法调用。这是因为周长和面积不是 Rect 类的属性。没有 getLength()方法和 getArea()方法，只有属性才能用 JSP 动作。

8.3.2　设置JavaBean的属性值

在例 8_3 中，通过对象调用 Rect 类中的 setHeight()方法和 setWidth()方法来设置属性值，但采用这种方式不能使 JSP 中静态和动态代码分离。为了解决这个问题，可以采用 JSP 动作<jsp:setProperty/>，该动作可以设置属性值。一般有 3 种方式。

1．通过表达式或字符串给JavaBean的属性赋值

通过表达式或字符串给 JavaBean 的属性赋值格式如下。

```
<jsp:setProperty name= "JavaBean 的对象名" property="属性名" value= "<%=b 表达式%>" />
```

或

```
<jsp:setProperty name="" property="" value=  字符串  />
```

采用表达式方式给 JavaBean 的属性赋值时，表达式的类型必须和属性的类型一致或是可以自动转换为属性的类型。如果采用字符串的方式赋值，则该字符串会自动转换为 JavaBean 中属性的类型。

例 8_4：设置 JavaBean 的属性值。

Book1.java

```
package pack;
public class Book
{
    String name;
    String author;
    int year,month,day;
    public Book()
    {
    }
    public void setName(String sname)
    {
        name=sname;
    }
    public void setAuthor(String sa)
    {
        author=sa;
    }
    public void setYear(int y)
    {
        year=y;
    }
    public void setMonth(int m)
    {
```

```
        month=m;
    }
    public void setDay(int d)
    {
        day=d;
    }
    public String getName()
    {
        try{
            byte b[]=name.getBytes("ISO-8859-1");
            name=new String(b);
            return name;
        }
        catch(Exception e)
        {
            return name;
        }
    }
    public String getAuthor()
    {
        try{
            byte b[]=author.getBytes("ISO-8859-1");
            author=new String(b);
            return author;
        }
        catch(Exception e)
        {
           return name;
        }
    }
    public int getYear()
    {
        return year;
    }
    public int getMonth()
    {
        return month;
    }
    public int getDay()
    {
        return day;
    }
}
```

Ex8_4.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312"%>
<%@ page import="pack.Book.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title></title>
</head>
<body>
    <%
        int y=2000;
        int m=10;
        int d=23;
    %>
    <jsp:useBean id="book1" class="pack.Book" scope="session">
    </jsp:useBean>
    <jsp:setProperty name="book1" property="name" value="基于 Java 的 Web 应用开发教程"/>
    <jsp:setProperty name="book1" property="author" value="孙璐"/>
    <jsp:setProperty name="book1" property="year" value="<%=y+8 %>"/>
    <jsp:setProperty name="book1" property="month" value="12"/>
    <jsp:setProperty name="book1" property="day" value="<%=d %>"/>
    <p> 图书的书名是：<jsp:getProperty name="book1" property="name"/>
    <p> 图书的作者是：<jsp:getProperty name="book1" property="author"/>
    <p> 图书的出版日期：<jsp:getProperty name="book1" property="year"/>年
    <jsp:getProperty name="book1" property="month"/>月
    <jsp:getProperty name="book1" property="day"/>日
</body>
</html>
```

程序的运行结果如图 8.3 所示。

图 8.3　通过表达式或字符串给 JavaBean 的属性赋值

在这个例子中，定义一个 Book 类，这是一个简单的图书信息 JavaBean，含有属性书名（name），作者（author），出版日期（year，month，day）。在 getName()和 getAuthor()

中分别对用户输入的汉字进行内码转换。这是由于在表单中输入的汉字的编码方式是 GB2312，而服务器的编码方式是 ISO-8859-1，所以需要进行编码转换。在 JSP 文件中，通过 JSP 动作<jsp:setProperty/>来设置属性值，设置属性值采用表达式和字符串的方式。通过 JSP 动作<jsp:getProperty/>来获取属性值。

<jsp:setProperty/>动作相当于调用 JavaBean 的 set 加属性名的方法。<jsp:getProperty/>动作相当于调用 JavaBean 的 get 加属性名的方法。在使用该动作之前，必须先调用 JSP 动作<jsp:useBean/>。

2．通过表单给JavaBean的属性赋值，表单的参数值和JavaBean的属性名相同

在通常情况下，JavaBean 的属性值大多都是通过表单由用户输入的，这时可以通过表单给 JavaBean 的属性赋值。在这种情况下，要求表单的参数值必须与 JavaBean 的属性名相同。JSP 会自动将用户在表单中输入的字符串类型的值转换成 JavaBean 的属性的类型。

例 8_5：根据例 8_4 中的 JavaBean，建立表单，完成赋值。

Ex8_5.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GB2312">
<title></title>
</head>
<body>
 <form action="Ex8_5.jsp" method="post">
    <p>图书书名：<input type="text" name="name">
    <p>图书作者：<input type="text" name="author">
    <p>出版日期：<input type="text" name="year" size="4" maxlength="4">
    <input type="text" name="month" size="4" maxlength="4">
    <input type="text" name="day" size="4" maxlength="4">
    <p><input type="submit" value="提交">
    <input type="reset" value="清除">
    <input type="button" value="退出" onclick="window.close()" >
 </form>
</body>
</html>
```

Ex8_5.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
     pageEncoding="gb2312"%>
```

```
<%@ page import="pack.Book.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>使用 setProperty 动作给属性赋值</title>
</head>
<body>
    <jsp:useBean id="book1" class="pack.Book" scope="page">
    </jsp:useBean>
    <jsp:setProperty name="book1" property="*" />
    <p>  图书的书名是：<jsp:getProperty name="book1" property="name"/>
    <p>  图书的作者是：<jsp:getProperty name="book1" property="author"/>
    <p>  图书的出版日期：<jsp:getProperty name="book1" property="year"/>年
    <jsp:getProperty name="book1" property="month"/>月
    <jsp:getProperty name="book1" property="day"/>日
    <p><input type="button" value="返回" onclick="history.back()" >
</body>
</html>
```

程序的运行结果如图 8.4 所示。

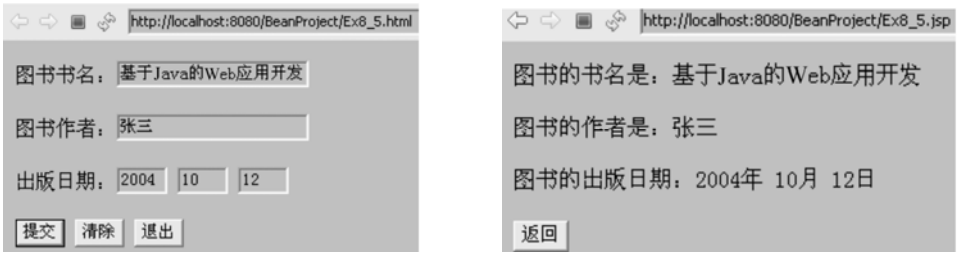


图 8.4　通过表单给 JavaBean 的属性赋值

在这个例子中，定义了一个 HTML 文件，在该文件中定义了一个表单，在表单中由用户输入多个属性的值。需要注意的是，表单中的各个文本框的 name 属性的值 name，author，year，month，day 必须和 JavaBean 的各个属性名相一致，这样 JSP 才可以对号入座，进行方法调用，完成赋值。在 JSP 文件中，首先用一个 JSP 动作<jsp:useBean/>来定义一个 JavaBean 的对象，然后通过 JSP 动作<jsp:setProperty/>给 JavaBean 的属性赋值。注意该动作中的 property 属性的值为“*”，表示赋的各个值由表单指定。最后通过多个 JSP 动作<jsp:getProperty/>来得到各个属性的值并输出。

3．通过表单给JavaBean的属性赋值，表单的参数值和JavaBean的属性名不同

例 8_6：根据例 8_4 中的 JavaBean，建立表单，完成赋值。

在例 8_5 中，表单中各个文本框的 name 属性的值和 JavaBean 的属性名相同，所以将 JSP 动作<jsp:setProperty/>中的 property 属性的值设为“*”。如果表单中各个文本框的 name 属性的值和 JavaBean 的属性名不相同，则不能应用例 8_5 的方法，因为 JSP 无法确定它们

的对应关系。这时就需要在<jsp:setProperty/>动作中多加一个参数来指出它们之间的对应关系。至于类型不同的问题，JSP 会自动进行转换。

Ex8_6.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GB2312">
<title></title>
</head>
<body>
 <form action="Ex8_6.jsp" method="post">
    <p>图书书名：<input type="text" name="sname">
    <p>图书作者：<input type="text" name="sauthor">
    <p>出版日期：<input type="text" name="syear" size="4" maxlength="4">
    <input type="text" name="smonth" size="4" maxlength="4">
    <input type="text" name="sday" size="4" maxlength="4">
    <p><input type="submit" value="提交">
    <input type="reset" value="清除">
    <input type="button" value="退出" onclick="window.close()" >
 </form>
</body>
</html>
```

Ex8_8.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
     pageEncoding="gb2312"%>
<%@ page import="pack.Book.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>使用 setProperty 动作给属性赋值</title>
</head>
<body>
    <jsp:useBean id="book1" class="pack.Book" scope="page">
    </jsp:useBean>
    <jsp:setProperty name="book1" property="name" param="sname" />
    <jsp:setProperty name="book1" property="author" param="sauthor"/>
    <jsp:setProperty name="book1" property="year" param="syear"/>
    <jsp:setProperty name="book1" property="month" param="smonth"/>
    <jsp:setProperty name="book1" property="day" param="sday"/>
    <p>  图书的书名是：<jsp:getProperty name="book1" property="name"/>
    <p>  图书的作者是：<jsp:getProperty name="book1" property="author"/>
```

```
        <p>  图书的出版日期：<jsp:getProperty name="book1" property="year"/>年
        <jsp:getProperty name="book1" property="month"/>月
        <jsp:getProperty name="book1" property="day"/>日
        <p><input type="button" value="返回" onclick="history.back()" >
    </body>
    </html>
```

程序的运行结果如图 8.5 所示。

图 8.5 通过表单给 JavaBean 的属性赋值

在这个例子中，在 HTML 中定义了 5 个文本框，name 属性的值分别为 sname，sauthor，syear，smonth，sday。而 JavaBean 的属性为 name，author，year，month，day，它们不相同，所以在<jsp:setProperty/>动作中必须指出在表单中哪个文本框中输入的信息是给哪个属性赋值。这就需要 property 属性和 param 属性。例如，<jsp:setProperty name="book1" property="name" param="sname" />表示 book1 对象的 name 属性的值由表单中的 sname 来指定。

8.4　JavaBean在项目中的应用

前面几节讲解了 JavaBean 的概念及用法，现在从人力资源管理系统比较简单的登录为例讲解 JavaBean 在项目中的具体应用。为了能够更加全面理解 JavaBean 在项目中的应用，需要将 JSP 及 Servlet 和与之对应的 JavaBean 结合使用讲解。

登录页面如图 6.7 所示。登录页面 login.jsp 的主要源代码见 6.5.1 节。

前面为视图层 login.jsp，现需要将用户名和密码传到服务器端，服务器再将数据传给 Servlet，Servlet 接收数据后进行业务逻辑处理。Servlet 所对应的代码如下。

```
package com.etop.usermanage.servlet;
import java.io.IOException;
import java.io.PrintWriter;
import java.math.BigDecimal;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.etop.bank.comm.pagebeanmapping.PageMap;
```

```
import com.etop.bank.usermanage.bean.UserLoginBean;
import com.etop.bank.usermanage.business.UserLoginDAOIMP;
import com.etop.bank.usermanage.dao.UserLoginDAO;
import com.etop.bank.util.tooles.bean.ToolesBean;
import com.etop.bank.util.tooles.business.Tooles;
public class UserLoginAction extends HttpServlet {
public String useraccount;
public String userpwd;
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
          useraccount=request.getParameter(“useraccount”);
          userpwd=request.getParameter(“userpwd”);
         UserLoginBean userbean=new UserLoginBean();
         userbean.setUseraccount(useraccount);
         userbean.setUserpwd(userpwd);
         UserLoginDAO userimp=UserLoginDAOIMP.getFactory().getImp(userbean);
         boolean flage=false;
         flage=userimp.userlogin();
         if(flage){
              String sql="select * from sd_sys_user";
              request.getSession().setAttribute("userbean", userbean);
              response.sendRedirect("../../bank/main.jsp");
                    }
         else{
        request.getSession().putValue("state", "用户名或密码输入错误，请重新输入！");
              response.sendRedirect("../../bank/userlogin/login.jsp");
         }
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
doGet(request,response);
    }
}
```

在上面的 Servlet 中，使用 UserLoginBean userbean=new UserLoginBean()语句，表示创建 JavaBean 类 UserLoginBean 对象 userbean。然后调用 set 方法将用户名和密码存入 JavaBean 中间载体中，即 userbean.setUseraccount(useraccount); userbean.setUserpwd(userpwd)。UserLoginBean 所对应的代码如下。

```
package com.etop.usermanage.bean;
import java.io.Serializable;
public class UserLoginBean implements Serializable {
    private int id;
```

```
        private String userid;
        private String useraccount ;
        private String userpwd ;
        private String groupid ;
        private String departmentid;
        private String positionid ;
        private String subbankid;
        private String usertype;
public String getDepartmentid()
{
return departmentid;
}
        public void setDepartmentid(String departmentid) {
                this.departmentid = departmentid;
        }
        public String getGroupid() {
                return groupid;
        }
        public void setGroupid(String groupid) {
                this.groupid = groupid;
        }
        public int getId() {
                return id;
        }
        public void setId(int string) {
                this.id = string;
        }
        public String getPositionid() {
                return positionid;
        }
        public void setPositionid(String positionid) {
                this.positionid = positionid;
        }
        public String getSubbankid() {
                return subbankid;
        }
        public void setSubbankid(String subbankid) {
                this.subbankid = subbankid;
        }
        public String getUseraccount() {
                return useraccount;
        }
        public void setUseraccount(String useraccount) {
                this.useraccount = useraccount;
```

```
        }
        public String getUserid() {
            return userid;
        }
        public void setUserid(String userid) {
            this.userid = userid;
        }
        public String getUserpwd() {
            return userpwd;
        }
        public void setUserpwd(String userpwd) {
            this.userpwd = userpwd;
        }
        public String getUsertype() {
            return usertype;
        }
        public void setUsertype(String usertype) {
            this.usertype = usertype;
        }
    }
```

为了能够全面理解 JavaBean 在整个项目中的位置，现将数据库代码写出，该数据的实现采用数据库访问对象 DAO 及 DAO 的实现。至于数据库的内部代码实现请参考第 9 章相关内容。

登录所对应的数据库访问对象 DAO 代码如下。

```
 package com.etop.bank.usermanage.dao;
public interface UserLoginDAO {
    /**
     *  用于验证用户登录的方法
     *  根据页面传递给 JavaBean（UserLoginBean）中的 useraccount 和 userpwd 的值，来与数据库中 sd_sys_user 表中的 useraccount 和 userpwd
     *  字段进行比较，如果都一致返回 true，否则返回 false
     * */
public boolean   userlogin();
}
```

前面定义了数据库访问对象 DAO，也就是数据操作的一个接口，接下来需要写一个类实现该接口以达到具体访问数据库的目的。实现该接口的代码如下。

```
package com.etop.usermanage.business;
import java.sql.ResultSet;
import java.sql.SQLException;
import com.etop.bank.comm.dbcom.DataBaseBsic;
import com.etop.bank.comm.dbcom.DataBaseBsicFactory;
import com.etop.bank.usermanage.bean.UserLoginBean;
import com.etop.bank.usermanage.dao.UserLoginDAO;
```

```
public final class UserLoginDAOIMP {
    private UserLoginDAOIMP() {
    }
    UserLoginImp userimp = new UserLoginImp();
    public static UserLoginDAOIMP getFactory() {
        return new UserLoginDAOIMP();
    }
    public UserLoginImp getImp(UserLoginBean userbean) {
        userimp.setUserbean(userbean);
        return userimp;
    }
    private final class UserLoginImp implements UserLoginDAO {
        private UserLoginImp() {
        }
        private UserLoginBean userbean = null;
      //获取数据库操作对象
        DataBaseBsic db = DataBaseBsicFactory.getFactory().getDBObject();
        boolean flage = true;
//验证用户登录
        public boolean userlogin() {
            String sql = "select    *    from sd_sys_user where    useraccount='"
                        + userbean.getUseraccount() + "' and userpwd='"
                        + userbean.getUserpwd() + "'";
          ResultSet rs=db.executeQuery(sql);
          try {
              if(rs==null||!rs.next()){
                      flage=false;
                  }
              else{
                  userbean.setDepartmentid(rs.getString("department_id"));
                  userbean.setGroupid(rs.getString("group_id"));
                  userbean.setSubbankid(rs.getString("subbank_id"));
                  userbean.setPositionid(rs.getString("position_id"));
                  userbean.setUserid(rs.getString("userid"));
                  userbean.setId(rs.getInt("id"));
                  userbean.setUseraccount(rs.getString("useraccount"));
                  userbean.setUsertype(rs.getString("user_type"));
                  userbean.setUserpwd("");
                  flage=true;
              }
          } catch (SQLException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
          }
```

```
            return flage;
        }
        public UserLoginBean getUserbean() {
            return userbean;
        }

        public void setUserbean(UserLoginBean userbean) {
            this.userbean = userbean;
        }
    }

}
```

在登录后台数据库实现当中，主要是验证输入的用户名和密码是否与从数据库查询出来的用户名和密码一致，采用方法为 public boolean userlogin()。在该方法中有如下一段代码。

```
String sql = "select * from sd_sys_user where useraccount='"+ userbean.getuseraccount() + "' and userpwd='"+ userbean.getUserpwd() + "'";
```

在该段代码中，主要是根据所输入的用户名和密码查询 sd_sys_user 表中的所有数据。而 userbean.getUseraccount()及 userbean.getUserpwd()方法是获取登录 Servlet 中通过调用 set 方法向 JavaBean 放入的值，代码如下。

```
UserLoginBean userbean=new UserLoginBean();
userbean.setUseraccount(useraccount);
userbean.setUserpwd(userpwd);
UserLoginDAO userimp=UserLoginDAOIMP.getFactory().getImp(userbean);
```

然后在后台数据库处理中，又调用 userbean.getUseraccount()及 userbean.getUserpwd()方法获取 JavaBean 对象 userbean 中的值。这样就达到了数据用户名和密码在 Servlet 与后台数据库实现传输的目的。

总之，无论数据是在 JSP 与 Servlet 之间传输还是在 Servlet 与后台数据库之间传输，JavaBean 都是通过 set 方法和 get 方法而作为中间载体达到数据传输目的的。

8.5　实训操作

在此次实训中，主要是要熟练掌握 JavaBean 的用法。现结合人力资源管理系统里的考核管理完成考核人员模块里的考核关系设置的功能。设置一次考核所需要的关系结构，包括考核名称管理、考核规则名称管理、考核项目管理、考核人设置。考核关系设置根据用户现行考核管理方法，确定出一次考核所有的基本数据项，如部门、职位、考核项、考核项权重、考核人、考核人权重等数据项。

习　　题

一、多项选择题

1．使用 JavaBean 的好处有哪些？（　　）

A．JavaBean 拥有标准化接口的优点，在开发期有可视化编程工具的支持，在运行期

有 JSP 和 J2EE 连接器的支持。

B．JavaBean 更明确地把 Web 页面的设计和软件的设计区分开来。

C．JavaBean 可以在多个应用程序中重用。

D．JavaBean 可以实现安全性、事务行为、并发性和持久性。

2．以下有关 JavaBean 的描述中，哪些是正确的？（　　）

A．JavaBean 是一个类。

B．如果属性为 xxx，则与属性有关方法为 getXxx()和 setXxx()。

C．对于布尔类型的属性，可以用 isXxx()方法。

D．类中可以没有不带参数的构造方法。

3．以下有关 JavaBean 使用的说法中，正确的是（　　）。

A．在 JSP 中使用 JavaBean 需要采用 JSP 动作<jsp:useBean/>。

B．在 JSP 中使用 JavaBean 需要采用 JSP 动作<jsp:useBean/>标签。

C．id 属性指出 JavaBean 的名字，即对象名。

D．class 属性指出 JavaBean 的类名。

4．以下有关 JavaBean 的描述中，正确的是（　　）。

A．JavaBean 的使用采用<jsp:useBean/>动作。

B．获取 JavaBean 的属性值采用<jsp:getProperty/>动作。

C．设置 JavaBean 的属性值采用<jsp:setProperty/>动作。

D．以上说法都正确。

5．以下是从 4 种不同的作用域中得到 JavaBean 的实例，说法错误的是（　　）。

A．page 是指在当前 Web 应用程序的所有 JSP 文件中取得实例，从 Page 对象中获取 JavaBean。

B. request 是指在当前的用户请求中取得实例，从 ServletRequest 对象中获取 JavaBean。

C．session 是指在当前的用户会话中取得实例，常用于一个用户登录在网站上全过程不同请求之间共享数据，从 HttpSession 对象中获取 JavaBean。

D．application 是指在当前的应用程序中取得实例，常用于不同用户访问同一个应用程序时共享数据，从 ServletContext 对象中获取 JavaBean 。

二、编程题

1. 创建 JavaBean，在 JavaBean 中包含 num 和 name 两个属性及各自的 get 和 set 方法，初始值为你的姓名和学号。创建 JSP 页面，调用 JavaBean 显示 num 和 name 属性的初始值，根据用户输入设置 num 和 name 属性，并显示设置后的属性值。

2．定义一个学生类的 JavaBean，包括姓名、性别、年龄、学号、成绩等属性。创建 JSP 页面，调用 JavaBean 设置各个属性的值，并显示其属性值。

第 9 章　数据库应用

📖 知识点

- 数据源
- JDBC-ODBC
- 关于数据库的查询、修改、添加、删除记录等操作
- 数据库连接池技术

📣 难点

- 数据库连接池
- 数据库的连接

✍ 掌握

- 关于数据库的各种操作
- 数据库连接池

任务引入

在 Java Web 项目开发中，离不开数据的存取，因此需要数据库对数据进行存储。那 Java 又通过怎样的方式与数据库建立连接而存取数据呢？Java 与数据库建立连接后又通过怎样的方式使得数据的存取效率大大提高呢？这就是本章需要解决的问题。

本章主要讲解 JDBC 的概念与执行原理及数据库连接池技术的运用。然后结合实际项目讲解数据库连接池在项目中的应用。

9.1　数据源与JDBC

9.1.1　JDBC概述

JDBC 是 Java 数据库连接（Java Database Connectivity）的简称。通过 JDBC，程序员能够用 Java 语言来访问任何关系型数据库，如 Oracle，SQL Server，Access，MySQL 等。在本书中，主要采用 SQL Server 数据库。使用 JDBC 能够执行 SQL 语句的查询、修改、添加、删除等操作。使用 JDBC 也能与多个数据源进行交互。

简单的说，JDBC 访问 SQL Server 数据库的步骤如下。

（1）建立数据库的连接。

（2）向数据库发送 SQL 语句。

（3）返回数据库处理结果。

在进行数据库连接时，主要采用两种方式。一种方式是通过 JDBC 驱动连接，对每一种数据库，不同的厂商都提供了连接数据库的 JDBC 驱动程序，可以利用该驱动程序进行连接。另一种方式是采用 JDBC-ODBC 桥接器进行连接。JDBC-ODBC 桥接器将所有的 JDBC 调用转换为 ODBC 调用，然后将其发送给 ODBC 驱动程序，再由 ODBC 驱动程序将调用

传送给数据库服务器。JDBC-ODBC 使得应用程序可以访问所有支持 ODBC 的数据库，即使这些数据库不支持 JDBC。

9.1.2　JDBC API

在进行数据库编程时，主要用到 java.sql 包中的类和接口。利用该包中的类和接口，可以将数据保存到数据库中，可以查询、修改、添加、删除数据库中的记录等。下面介绍几个重要的类和接口。

1．DriverManage类

DriverManage 类是用于管理 JDBC 驱动程序的类。它在数据库和相应驱动程序之间建立连接。另外，DriverManage 类也处理诸如驱动程序登录时间限制及登录和跟踪消息的显示等事务。主要用途是通过 getConnection()方法来获得 Connection 对象的引用。其方法格式如下。

```
static Connection getConnection(String URL);        //连接指定的库
static Connection getConnection(String URL, String user,String password); //连接指定的库，使用用户名 user 和密码 password
```

其中数据库连接标志 URL 的模式如下。

```
jdbc:<subprotocol>:<subname>                    //协议：子协议：子名
```

协议只有 jdbc 一种；子协议被用来识别数据库驱动程序，不同的数据库子协议不同；不同的数据库驱动程序采用不同的子名，不同的专门驱动程序可以采用不同的实现。

JDBC-ODBC 对应的 URL 如下。

```
jdbc:odbc:数据库别名
```

SQL Server 数据库对应的 URL 如下。

```
jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=数据库名称
```

Oracle 数据库对应的 URL 如下。

```
jdbc:db2:数据库名
```

2．Driver接口

使用该接口加载驱动程序。

如果使用 JDBC-ODBC 桥接器，加载方式如下。

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDreiver");
```

如果使用 JDBC 加载 SQL Server 驱动，方式如下。

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

如果使用 JDBC 加载 Oracle 驱动，方式如下。

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

如果使用 JDBC 加载 DB2 驱动，方式如下。

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
```

3．Connection接口

Connection 对象是通过 DriverManage.getConnection()方法获得的，表示驱动程序提供的与数据库连接的对话。其常用方法如下。

Statement createStatement()，创建 SQL 语句。

PreparedStatement prepareStatement(String sql)，把 SQL 语句提交到数据库进行预编译。

CallableStatement prepareCall(String sql)，处理存储过程。

4．Statement接口

Statement 对象用于执行 SQL 语句，语句包括查询、修改、添加、删除等。实际上有 3 种 Statement 对象：Statement、PreparedStatement（它从 Statement 继承而来）和 Callable Statement（它从 PreparedStatement 继承而来）。它们都用于在给定连接上执行 SQL 语句。Statement 对象用于执行不带参数的简单 SQL 语句，PreparedStatement 对象用于执行带或不带 in 参数的预编译 SQL 语句，CallableStatement 对象用于执行对数据库存储过程的调用。

Statement 的常用方法如下。

ResultSet executeQuery(String sql)，执行查询语句，返回 ResultSet 对象。

int executeUpdate(String sql)，执行更新操作，返回更新的行数。

boolean execute(String sql)，执行一个修改、添加或删除语句，返回的布尔值表示语句是否被执行成功。

当建立了到某个数据库的连接之后，就可用该连接发送 SQL 语句。Stratement 对象用 Connection 的方法 createStatement()创建。例如：

```
Connection conn=DriverManager.getConnection("jdbc:odbc:test");
Statement stmt=conn.createStatement();
ResultSet rs=stmt.executeQuery("select * from student ");
```

5．ResultSet接口

该对象保存了执行查询数据库语句后产生结果的一个集合。其常用方法如下。

boolean next()，把当前的指针向下移动一位。

void close()，释放 ResultSet 对象资源。

boolean absolute(int rows)，将结果集合移动到指定行。

getXXX(String column)或 getXXX(int column)，获得字段值，XXX 可以为 Byte，Date，Float，Double，Boolean，Object，Blob，Clob，String，Int，Long 等类型。当参数为整型时，表示列数，第 1 列为 1，第 2 列为 2……

9.1.3　在JSP或Servlet中访问SQL Server数据库的步骤

在 JSP 或 Servlet 中访问 SQL Server 数据库可以采用两种方式。下面分别就这两种方式来讲解访问步骤。

1．通过JDBC驱动进行访问步骤

利用微软的 Microsoft SQL Server Driver for JDBC 驱动程序，可以连接 SQL Server 数据库。使用时需要首先到微软的网站上下载驱动程序。下载完后进行驱动程序的安装。安装完后，需要复制驱动程序安装目录 lib 文件夹下的 3 个 jar 文件到 Tomcat 安装目录的 common\lib 文件夹下，或者把这 3 个 jar 文件的目录设置到环境变量 CLASSPATH 中。同时需要在项目中导入这 3 个 jar 文件，方法为：在 MyElipse 中的项目上单击鼠标右键，打开“Build Path”弹出式菜单，选择“Add External Archives”选项，打开“JAR Selected”对话框，选中这 3 个 jar 文件，单击“打开”按钮即可。必要时可以安装 SQL Server SP4 补丁。

在程序中访问数据库的步骤如下。

（1）导入与数据库相关的类。

```
import java.sql.*;
```

（2）加载数据库驱动程序。

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
```

（3）建立数据库连接。

```
String URL="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=数据库名称";
String user= "数据库用户名";
String password= "数据库密码";
Connection conn=DriverManage.getConnection(URL,user,password);
```

（4）创建 Statement 对象。

```
Statement stmt=conn.createStatement();
```

（5）执行 SQL 语句。

```
ResultSet rs=stmt.executeQuery("SQL 语句");      //执行查询语句
ResultSet rs=stmt.executeUpdate("SQL 语句");     //执行修改、添加或删除语句
```

（6）处理结果。

例 9_1：通过 JDBC 进行连接，访问 BOOK 数据库中的 bookinfo 表中的所有记录。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312"%>
<%@ page import="java.sql.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>采用 JDBC 查询 SQL Server 数据库</title>
</head>
```

```
<body>
  <%
    Connection conn;
    Statement stmt;
    ResultSet rs;
    try{
          String URL ="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=BOOK";
          String user="sa";
          String password="sa";
          Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
          out.println("a");
          conn = DriverManager.getConnection(URL,user,password);
          out.println("b");
          stmt=conn.createStatement();
          rs=stmt.executeQuery("select * from bookinfo");
          out.print("<table border>");
          out.print("<tr>");
            out.print("<th width=100>"+"编号");
            out.print("<th width=100>"+"书名");
            out.print("<th width=100>"+"作者");
            out.print("<th width=100>"+"出版社");
            out.print("<th width=100>"+"出版日期");
            out.print("<th width=100>"+"单价");
            out.print("<th width=100>"+"图书分类");
          out.print("</tr>");
          while(rs.next())
          {
          out.print("<tr>");
            out.print("<td >"+rs.getString(1)+"</td>");
            out.print("<td >"+rs.getString("name")+"</td>");
            out.print("<td>"+rs.getString("author")+"</td>");
            out.print("<td>"+rs.getString(4)+"</td>");
            out.print("<td >"+rs.getString("time")+"</td>");
            out.print("<td>"+rs.getFloat(6)+"</td>");
            out.print("<td>"+rs.getString(7)+"</td>");
          out.print("</tr>");
          }
       }
    catch(ClassNotFoundException e1){out.println("ddd");}
    catch(SQLException e2){out.println("eee");e2.getMessage();}
  %>
</body>
    </html>
```

2．通过JDBC-ODBC桥接器进行访问步骤。

通过 JDBC-ODBC 桥接器访问 SQL Server 数据库的步骤是：首先建立数据库和表，其次创建 DSN 数据源，最后编写数据库访问程序。

创建 DSN 数据源的过程如下。首先双击“控制面板”中的“管理工具”，再双击“数据源 ODBC”，打开数据源管理器。选择“系统 DSN”选项卡，然后单击“添加”按钮，打开“创建新数据源”对话框，如图 9.1 所示。这时选择所要安装数据源的数据库类型，这里选择“SQL Server”，单击“完成”按钮，打开“建立新的数据源到 SQL Server”对话框，如图 9.2 所示。输入数据源的名称，这里输入“book”，选择数据库服务器，如果用户安装了 SQL Server，则会自动产生服务器的名字，单击“下一步”按钮，打开“SQL Server 应该如何验证登录 ID 的真伪”对话框。用户可以根据安装 SQL Server 时的选项来选择，单击“下一步”按钮，选择“更改默认的数据库为（D）：”复选框，如图 9.3 所示。选择新建的数据库“BOOK”，单击“下一步”按钮，然后单击“完成”按钮。此时显示“ODBC Microsoft SQL Server 安装”对话框，单击“测试数据源”按钮，此时若显示“测试成功”，则表示创建数据源成功。单击“确定”按钮，关闭对话框。这时在“系统 DSN”选项卡上将会出现 BOOK 数据源。

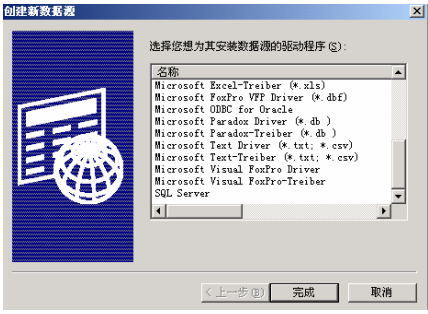


图 9.1　创建新数据源

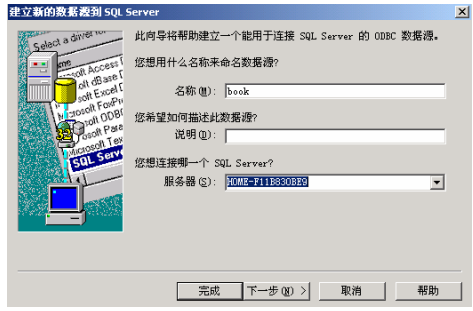


图 9.2　建立新的数据源到 SQL Server

图 9.3　选择新建的数据库

最后要编写数据库访问程序。通过 JDBC-ODBC 桥接器进行 SQL Server 数据库连接时需要用到驱动程序“sun.jdbc.odbc.JdbcOdbcDriver”，该驱动程序不需要另外加载，Tomcat 服务器自带这个驱动程序。在程序中访问数据库的步骤如下。

（1）导入与数据库相关的类。

```
import java.sql.*;
```

（2）加载数据库驱动程序。

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDreiver");
```

（3）建立数据库连接。

```
Connection conn=DriverManage.getConnection("jdbc:odbc:数据库别名");
```

（4）创建 Statement 对象。

```
Statement stmt=conn.createStatement();
```

（5）执行 SQL 语句。

```
ResultSet rs=stmt.executeQuery("SQL 语句");          //执行查询语句
ResultSet rs=stmt.executeUpdate("SQL 语句");         //执行修改、添加或删除语句
```

（6）处理结果。

例 9_2：通过 JDBC-ODBC 桥接器访问 BOOK 数据库中的 bookinfo 表，显示表中的所有记录。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312"%>
<%@ page import="java.sql.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>采用 JDBC-ODBC 桥接器查询 SQL Server 数据库</title>
</head>
<body>
  <%
    Connection conn;
    Statement stmt;
    ResultSet rs;
    try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        conn=DriverManager.getConnection("jdbc:odbc:book");
        stmt=conn.createStatement();
        rs=stmt.executeQuery("select * from bookinfo");
        out.print("<table border>");
        out.print("<tr>");
          out.print("<th>"+"编号");
          out.print("<th >"+"书名");
          out.print("<th >"+"作者");
          out.print("<th >"+"出版社");
          out.print("<th >"+"出版日期");
          out.print("<th >"+"单价");
```

```
                out.print("<th >"+"图书分类");
              out.print("</tr>");
              while(rs.next())
              {
              out.print("<tr>");
                out.print("<td >"+rs.getString(1)+"</td>");
                out.print("<td >"+rs.getString("name")+"</td>");
                out.print("<td>"+rs.getString("author")+"</td>");
                out.print("<td>"+rs.getString(4)+"</td>");
                out.print("<td >"+rs.getString("time")+"</td>");
                out.print("<td>"+rs.getFloat(6)+"</td>");
                out.print("<td>"+rs.getString(7)+"</td>");
              out.print("</tr>");
              }
            }
        catch(ClassNotFoundException e1){}
        catch(SQLException e2){}
      %>
    </body>
    </html>
```

在这两个程序中，是按表中的顺序依次查询表中的记录。用到了 next()方法，将当前记录顺序下移。

由于使用 SQL 语句可以执行各种不同的查询操作，下面就描述各种不同的查询方式。

例 9_1 和例 9_2 程序的运行结果相同，如图 9.4 所示。

编号	书名	作者	出版社	出版日期	单价	图书分类
1001	JSP	张三	清华大学出版社	2008-03-12 00:00:00.000	23.0	计算机
1002	Java程序设计	李四	电子工业出版社	2002-02-12 00:00:00.000	30.0	计算机
1003	大学英语	王朝	外文出版社	2000-03-23 00:00:00.000	34.0	外文
1004	DSP技术	倪言	电子工业出版社	2001-04-23 00:00:00.000	45.0	电子
1005	数据结构	张三	机械工业出版社	1998-02-21 00:00:00.000	25.0	计算机
1006	数字电子技术	王向红	清华大学出版社	2003-07-21 00:00:00.000	43.0	电子

图 9.4　查询 bookinfo 表中的所有记录

9.2　查询记录

9.2.1　根据条件查询记录

根据条件查询记录可以采用 SQL 语句中的 where 子句来实现。where 子句被用来决定哪些行在查询结果中，哪些行应该删除。使用 where 子句应注意，对于所有运算符，在检验数字数据时，不必使用单引号将被比较的数据括起来。

在例 9_3 中，采用 JavaBean＋JSP＋HTML 的形式。JavaBean 来完成数据库的连接，JSP 来完成具体的查询操作，HTML 完成由用户给出查询条件。

例 9_3：根据图书价格和出版社进行查询。

DatabaseConn.java

```
package pack;
import java.sql.*;
public class DatabaseConn
{
    Connection conn = null;
    public DatabaseConn()
    {
      try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      conn=DriverManager.getConnection("jdbc:odbc:book","sa","sa");
      }
      catch(ClassNotFoundException e1){}
      catch(SQLException e2){}
    }
    //取得数据库连接的方法
    public Connection getConn()
    {
      System.out.print("success");
      return conn;
    }
}
```

Ex9_3.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>查询条件</title>
</head>
<body>
  <form action="Ex9_3.jsp">
  请输入查询条件：<p>
      单价：<input type="text" name="price" value="0" >以上
    <p>出版社：
    <select name="publish">
        <option value="1">清华大学出版社
        <option value="2">电子工业出版社
        <option value="3">外文出版社
    </select>
    <p>
    <input type="submit" name="submit" value="提交">
```

```
            <input type="reset" name="reset" value="重置">
            <input type="button" name="close" value="退出" onclick=window.close()>
        </form>
    </body>
        </html>
```

Ex9_3.jsp

```
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
    <html>
    <head>
    <%@ page language="java" contentType="text/html; charset=gb2312"
          pageEncoding="gb2312"%>
    <%@ page import="java.sql.*" %>
    <meta http-equiv="Content-Type" content="text/html; charset=gb2312">
    <title>按条件查询</title>
    </head>
    <body>
    <jsp:useBean id="dataBean" class="pack.DatabaseConn" />
    <center>
    <h3>JSP 应用数据库连接 JavaBean 查询表中记录</h3>
    <%
            String pub="";
            String money=request.getParameter("price");
            String p=request.getParameter("publish");
            String condition=" ";
            if(p.equals("1")){
               pub="清华大学出版社";
            }
            else if(p.equals("2"))    {
            pub="电子工业出版社";
            }
            else if(p.equals("3")){
            pub="外文出版社";
            }
            condition="price>="+money+" and "+"publisher= "+"'"+pub+"'";

       try{
            Connection conn = dataBean.getConn();
            if(conn != null)
            {
               out.println("<p>成功取得 JavaBean 的数据库连接.</p>");
            Statement stmt=conn.createStatement();
            String sql="select * from bookinfo where "+condition;
            ResultSet rs=stmt.executeQuery(sql);
```

```
                out.print("<table border>");
                    out.print("<tr>");
                        out.print("<th>"+"编号");
                        out.print("<th>"+"书名");
                        out.print("<th>"+"作者");
                        out.print("<th >"+"出版社");
                        out.print("<th >"+"出版日期");
                        out.print("<th >"+"单价");
                        out.print("<th >"+"图书分类");
                        out.print("</tr>");
                        while(rs.next())
                        {
                             out.print("<tr>");
                            out.print("<td >"+rs.getString(1)+"</td>");
                            out.print("<td >"+rs.getString("name")+"</td>");
                            out.print("<td>"+rs.getString("author")+"</td>");
                            out.print("<td>"+rs.getString(4)+"</td>");
                            out.print("<td >"+rs.getString("time")+"</td>");
                            out.print("<td>"+rs.getFloat(6)+"</td>");
                            out.print("<td>"+rs.getString(7)+"</td>");
                          out.print("</tr>");
                          }
                     }
            }
            catch(SQLException e2){};
        %>
        <p>
            <input type="button" name="btn" value="返回" onclick="history.back()">
        </body>
        </html>
```

程序的运行结果如图 9.5 所示。

在例 9_3 中，用户首先在 HTML 文件中输入需要查询的价格（在多少元以上），同时输入需查询的出版社名称。单击“提交”按钮后提交给 Ex9_3.jsp 文件，在该文件中，首先利用 JavaBean 进行数据库的连接，然后得到 HTML 中价格和出版社的值，根据查询条件进行 SQL 语句查询。需要注意的是，查询的 SQL 语句如下。

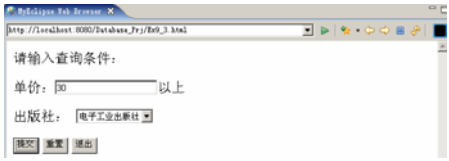


图 9.5　根据条件查询

```
condition="price>="+money+" and "+"publisher= "+"'"+pub+"'";
String sql="select * from bookinfo where "+condition;
```

where 子句用来决定查询条件。注意：语句中 where 后面一定要有空格；查询条件 condition 变量中，每一部分条件用双引号括起来，如“price>=”、“publisher=”和“and”，and 前后必须要有空格；查询条件的参数值如果是字符串，则必须要用单引号括起来，如果是整型变量，则不需要用单引号括起来。

9.2.2　对查询的记录进行排序输出

关系数据库的一个特点是表中列和行的顺序并不重要。数据库也不按顺序来处理它们。这就意味着随便按照哪种顺序从数据库中检索记录都很简单。一般来说，按照它们最初被插入的顺序来返回查询结果。在查看查询结果时，可能愿意看到按照具体某列内容排序输出的结果。Order by 子句就是被用来对查询结果排序。

例 9_4：对 bookinfo 数据库表按价格进行排序。

DatabaseConn.java 与例 9_3 相同。

Ex9_4.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>对查询结果进行排序</title>
</head>
<body>
    <form action="Ex9_4.jsp" method="post">
        <p>选择排序方式：</p>
        <input type="radio" name="sort" value="price">价格
        <input type="radio" name="sort" value="time">出版日期
        <input type="radio" name="sort" value="publisher">出版社
        <p>选择升序或降序：</p>
        <input type="radio" name="mode" value="ASC" >升序
        <input type="radio" name="mode" value="DESC" >降序
        <P>
        <input type="submit" name="submit" value="提交">
```

```
            <input type="reset" name="reset" value="重置">
            <input type="button" name="close" value="退出" onclick="window.close()">
        </form>
    </body>
        </html>
```

Ex9_4.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
      pageEncoding="gb2312"%>
<%@ page import="java.sql.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>对查询结果进行排序</title>
</head>
<body>
    <jsp:useBean id="dataBean" class="pack.DatabaseConn" />
<center>
<h3>JSP 应用数据库连接 JavaBean 查询表中记录</h3>
<%
        String sort=request.getParameter("sort");
        String mode=request.getParameter("mode");
   try{
            Connection conn = dataBean.getConn();
            if(conn != null)
            {
               out.println("<p>成功取得 JavaBean 的数据库连接.</p>");
        Statement stmt=conn.createStatement();
        String sql="select * from bookinfo order by "+sort+" "+mode;
        ResultSet rs=stmt.executeQuery(sql);
        out.print("<table border>");
            out.print("<tr>");
                out.print("<th>"+"编号");
                out.print("<th>"+"书名");
                out.print("<th>"+"作者");
                out.print("<th >"+"出版社");
                out.print("<th >"+"出版日期");
            out.print("<th >"+"单价");
            out.print("<th >"+"图书分类");
            out.print("</tr>");
            while(rs.next())
            {
                out.print("<tr>");
```

```
                out.print("<td >"+rs.getString(1)+"</td>");
                out.print("<td >"+rs.getString("name")+"</td>");
                out.print("<td>"+rs.getString("author")+"</td>");
                out.print("<td>"+rs.getString(4)+"</td>");
                out.print("<td >"+rs.getString("time")+"</td>");
                out.print("<td>"+rs.getFloat(6)+"</td>");
                out.print("<td>"+rs.getString(7)+"</td>");
            out.print("</tr>");
            }
          }
    }
    catch(SQLException e2){};
%>
<p><center>
<input type="button" name="btn" value="返回" onclick="history.back()">
</center>
</body>
</html>
```

程序的运行结果如图 9.6 所示。

图 9.6　对查询结果进行排序

在例 9_4 中，仍然采用 JavaBean＋JSP＋HTML 的形式。

9.2.3　通配符查询

通配符查询可以通过 like 子句来实现。可用 like 子句来创建与字符串模式相匹配的简单表达式。在需要查找具有某些相同内容的字符串或已知字符串的一部分时，可以用它来查询到需要的字符串。like 子句也可以用 not 运算符来求反。

SQL 标准对于模式匹配表达式提供了两种通配符：“%”和“_”。使用“%”代替一个

或多个字符，使用下画线“_”代替一个字符。

例 9_5：对 bookinfo 数据库表查询出版社名称中含有“工业”两个字的记录。

Ex9_5.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%@ page import="java.sql.*" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+
path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>通配符查询</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->

  </head>

  <body>
<%
    Connection conn;
    Statement stmt;
    ResultSet rs;
    try{
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
        conn                                                                    =
DriverManager.getConnection("jdbc:sqlserver://localhost:1433;DatabaseName=BOOK",
        "sa","sa");
        stmt=conn.createStatement();
         rs=stmt.executeQuery("select * from bookinfo where publisher like '%工业%' ");
         out.print("<table border>");
```

```
                out.print("<tr>");
                   out.print("<th>"+"编号");
                   out.print("<th>"+"书名");
                   out.print("<th>"+"作者");
                   out.print("<th >"+"出版社");
                   out.print("<th >"+"出版日期");
                   out.print("<th >"+"单价");
                   out.print("<th >"+"图书分类");
                out.print("</tr>");
                while(rs.next())
                {
                out.print("<tr>");
                   out.print("<td >"+rs.getString(1)+"</td>");
                   out.print("<td >"+rs.getString("name")+"</td>");
                   out.print("<td>"+rs.getString("author")+"</td>");
                   out.print("<td>"+rs.getString(4)+"</td>");
                   out.print("<td >"+rs.getString("time")+"</td>");
                   out.print("<td>"+rs.getFloat(6)+"</td>");
                   out.print("<td>"+rs.getString(7)+"</td>");
                out.print("</tr>");
                }
              }
            catch(ClassNotFoundException e1){}
            catch(SQLException e2){out.println(e2);}
           %>
           </body>
        </html>
```

程序的运行结果如图 9.7 所示。

图 9.7　通配符查询

在该程序中，是查询出版社名称中含有“工业”两个字的所有记录。SQL 语句为“select * from bookinfo where publisher like '%工业%'”。“工业”前后有“%”代表“工业”两个字的前面和后面可以有多个字符。如果把 SQL 语句改为“select * from bookinfo where publisher like '_工业_'”，则程序的运行结果中没有任何记录，因为下画线只代表一个字符。

9.2.4　prepareStatement()方法的应用

在进行条件查询中，查询条件的参数值可以先用“？”代替，随后再设置其值，这样可以使 SQL 语句更加清晰明了，而且可以减少书写错误。它的优势在多条件查询中尤为突出。下面修改例 9_3 中的 Ex9_3.jsp 程序来体会这种方式的用法。

例 9_6：prepareStatement()方法的应用。

Ex9_6.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312"%>
<%@ page import="java.sql.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>按条件查询</title>
</head>
<body>
<jsp:useBean id="dataBean" class="pack.DatabaseConn" />
<center>
<h3>JSP 应用数据库连接 JavaBean 查询表中记录</h3>
<%
        String pub="";
        String money=request.getParameter("price");
        String p=request.getParameter("publish");
        float m=Float.parseFloat(money);
        System.out.println(m);
        if(p.equals("1"))
        { pub="清华大学出版社"; }
        else if(p.equals("2"))
        {   pub="电子工业出版社";          }
        else if(p.equals("3"))
        {   pub="外文出版社";          }
  try{
         Connection conn = dataBean.getConn();
         if(conn != null)
         {
            out.println("<p>成功取得 JavaBean 的数据库连接.</p>");
        String sql="select * from bookinfo where price>=? and publisher=?";
        PreparedStatement ps=conn.prepareStatement(sql);
        ps.setFloat(1,m);
        ps.setString(2,pub);
        ResultSet rs=ps.executeQuery();
        out.print("<table border>");
```

```
                out.print("<tr>");
                  out.print("<th>"+"编号");
                  out.print("<th>"+"书名");
                  out.print("<th>"+"作者");
                  out.print("<th >"+"出版社");
                  out.print("<th >"+"出版日期");
                 out.print("<th >"+"单价");
                 out.print("<th >"+"图书分类");
                 out.print("</tr>");
                 while(rs.next())
                 {
                     out.print("<tr>");
                    out.print("<td >"+rs.getString(1)+"</td>");
                    out.print("<td >"+rs.getString("name")+"</td>");
                    out.print("<td>"+rs.getString("author")+"</td>");
                    out.print("<td>"+rs.getString(4)+"</td>");
                    out.print("<td >"+rs.getString("time")+"</td>");
                    out.print("<td>"+rs.getFloat(6)+"</td>");
                    out.print("<td>"+rs.getString(7)+"</td>");
                 out.print("</tr>");
                 }
               }
     }
     catch(SQLException e2){};

%>
<p><center>
<input type="button" name="btn" value="返回" onclick="history.back()">
</center>
</body>
</html>
```

在该程序中，SQL 语句的查询条件的参数值用“？”代替。首先通过 prepareStatement（String sql）方法把 SQL 语句提交到数据库进行预编译，然后调用 PreparedStatement 的方法对象设置“？”的值。注意：根据表中字段类型的不同调用不同的“set+类型”方法。在该例中，price 字段类型为货币类型，而得到用户输入的值为字符串类型，需要把它转换成 float 类型，调用 setFloat（1，m）方法给 price 参数赋值。在 SQL 中参数的下标是从 1 开始的，这点需要大家注意。

9.3　修改记录

修改记录实际上是对数据库表的某一条记录的某些字段进行修改。修改记录是使用 Update 语句来实现的。其一般格式如下。

```
String sql= "update 表名 set 字段名 1＝参数值 1，字段名 2＝参数值 2 ….where 字段名＝
参数值"
Statement stmt=null;
stmt.executeUpdate( sql );
```

Update 语句一共有 3 部分：第 1 部分指出要修改的是哪一张表；第 2 部分是 set 子句，指出其中要修改的列和要插入的值；第 3 部分是 where 子句，指定表中哪些行将要被修改。如果只对表中的某一行进行修改，则在 where 子句使用主关键字来做条件，因为关键字的值是唯一的。如果没有 where 子句，则会修改表中的所有记录。

修改的 SQL 语句通过 Statement 对象的 executeUpdate（String sql）方法来完成数据库的修改操作。

例 9_7：修改 bookinfo 数据库表中的记录。

Ex9_7.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Ex9_7.html</title>
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="this is my page">
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <!--<link rel="stylesheet" type="text/css" href="./styles.css">-->
  </head>
  <body>
  <form action="Ex9_7_1.jsp" >
        请输入需要修改的图书的编号：
    <select name="no">
      <option value="1001">1001</option>
      <option value="1002">1002</option>
      <option value="1003">1003</option>
      <option value="1004">1004</option>
      <option value="1005">1005</option>
      <option value="1006">1006</option>
    </select>
    <p><center><input type="submit" name="submit" value="提交">
    <input type="reset" name="reset" value="重置">
  </form>
  </body>
</html>
```

Ex9_7_1.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
```

```
<%@ page import="java.sql.*" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+
path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>显示需修改的记录</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->
  </head>
<body>
  <form action="Ex9_7_2.jsp">
  <%
    Connection conn;
    Statement stmt;
    ResultSet rs;
    String sno=request.getParameter("no");
    System.out.println("no="+sno);
    try{
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
        conn = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;DatabaseName=BOOK",
        "sa","sa");
        stmt=conn.createStatement();
        rs=stmt.executeQuery("select * from bookinfo where no='"+sno+"'");
        while(rs.next())
        {
            String name=rs.getString("name");
            String author=rs.getString("author");
            String pub=rs.getString(4);
            String time=rs.getString("time");
            float price=rs.getFloat(6);
            String category=rs.getString(7);
    %>
      <p>修改的值为：
```

```
        <input type="hidden" name="no" value="<%=sno %>">
        <br>书名：<input type="text" name="name" value="<%=name%>">
        <br>作者：<input type="text" name="author" value="<%=author %>">
        <br>出版社：<input type="text" name="pub" value="<%=pub %>">
        <br>出版日期：<input type="text" name="date" value="<%=time %>">
        <br>价格：<input type="text" name="price" value="<%=price %>">
        <br>分类：<input type="text" name="category" value="<%=category %>">
        <p><input type="submit" name="submit" value="修改记录">
        <input type="reset" name="reset" value="取消修改">
      </form>
    <%
            }
    }
    catch(ClassNotFoundException e1){}
    catch(SQLException e2){out.println(e2);}
  %>
  </body>
</html>
```

Ex9_7_2.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=gb2312"
    pageEncoding="gb2312"%>
<%@ page import="java.sql.*" %>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>显示修改后的记录</title>
</head>
<body>
  修改后表中的记录为：
  <%
    Connection conn;
    Statement stmt;
    ResultSet rs;
    String sno=request.getParameter("no");
    String name=request.getParameter("name");
    name=new String(name.getBytes("ISO8859-1"),"GB2312");
    String author=request.getParameter("author");
    author=new String(author.getBytes("ISO8859-1"),"GB2312");
    String pub=request.getParameter("pub");
    pub=new String(pub.getBytes("ISO8859-1"),"GB2312");
    String time=request.getParameter("date");
    String money=request.getParameter("price");
```

```
    String category=request.getParameter("category");
    category=new String(category.getBytes("ISO8859-1"),"GB2312");
    float m=Float.parseFloat(money);
    try{
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
        conn = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;DatabaseName=BOOK",
        "sa","sa");
        stmt=conn.createStatement();
        String sql1="update bookinfo set name='"+name+"' where no='"+sno+"'";
        String sql2="update bookinfo set author='"+author+"' where no='"+sno+"'";
        String sql3="update bookinfo set publisher='"+pub+"' where no='"+sno+"'";
        String sql4="update bookinfo set time='"+time+"' where no='"+sno+"'";
        String sql5="update bookinfo set price="+m+" where no='"+sno+"'";
        String sql6="update bookinfo set category='"+category+"' where no='"+sno+"'";
        stmt.executeUpdate(sql1);
        stmt.executeUpdate(sql2);
        stmt.executeUpdate(sql3);
        stmt.executeUpdate(sql4);
        stmt.executeUpdate(sql5);
        stmt.executeUpdate(sql6);
        System.out.println("success");
        rs=stmt.executeQuery("select * from bookinfo");
        out.print("<table border>");
        out.print("<tr>");
          out.print("<th>"+"编号");
          out.print("<th>"+"书名");
          out.print("<th>"+"作者");
          out.print("<th >"+"出版社");
          out.print("<th >"+"出版日期");
          out.print("<th >"+"单价");
          out.print("<th >"+"图书分类");
        out.print("</tr>");
        while(rs.next())
        {
        out.print("<tr>");
          out.print("<td >"+rs.getString(1)+"</td>");
          out.print("<td >"+rs.getString("name")+"</td>");
          out.print("<td>"+rs.getString("author")+"</td>");
          out.print("<td>"+rs.getString(4)+"</td>");
          out.print("<td >"+rs.getString("time")+"</td>");
          out.print("<td>"+rs.getFloat(6)+"</td>");
          out.print("<td>"+rs.getString(7)+"</td>");
         out.print("</tr>");
        }
```

```
            }
        catch(ClassNotFoundException e1){}
        catch(SQLException e2){out.println(e2);}
      %>
    </body>
    </html>
```

首先选择图书的编号，单击“提交”按钮提交给 Ex9_7_1.jsp 页面进行处理。在该文件中，根据传递过来的图书编号在 bookinfo 表中查询该图书的信息，然后显示，等待用户的修改。当用户单击“修改记录”按钮提交记录的值后，程序转去 Ex9_7_2.jsp 页面显示修改后表中的所有记录。程序的运行结果图如图 9.8 所示。

图 9.8　修改记录

9.4　添加记录

添加记录实际上是向数据库表的最后一行插入记录。添加记录的 SQL 语句格式如下。

```
String sql= "insert into  表名  values (  参数 1，参数 2，…参数 n ) ";
Statement stmt;
stmt.executeUpdate ( sql );
```

在以上 SQL 语句中，圆括号内参数的个数和表中字段的个数相同，参数的顺序和表中字段的顺序相同。添加记录的 SQL 语句实际上是根据圆括号内参数的顺序依次给表中各个字段进行赋值的。

如果添加记录时，只给部分字段赋值，或参数的顺序和字段的顺序不同时，可以采用如下格式。

```
String sql="insert into  表名(字段 1，字段 2，…字段 n ) values (  参数 1，参数 2，…参数 n ) ";
```

在这种格式中，参数 1 的值被赋给字段 1，参数 2 的值被赋给字段 2，以此类推。

如果表中的字段为字符型或日期型，则参数应用单引号括起来；如果表中的字段为其他类型（如 int 型成货币型），则不需要用单引号括起来。

例 9_8：给 bookinfo 数据库表添加一条记录。

Ex9_8.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>添加记录</title>
</head>
  <body>
    <form action="Ex9_8.jsp">
      <p>  请输入需要添加记录的值：
      <p>
       编    号：
       <input type="text" size="50" name="no" value="0"> <br>
       书    名：
       <input type="text" size="50" name="name" ><br>
       作    者：
       <input type="text" size="50" name="author" ><br>
       出 版 社：
       <input type="text" size="50" name="pub" ><br>
       出版日期：
       <input type="text" size="50" name="date" ><br>
       价    格：
       <input type="text" size="50" name="money" ><br>
       分    类：
       <input type="text" size="50" name="cat" ><br>
      <p>
        <input type="submit" name="submit" value="添加记录">
        <input type="reset" name="reset" value="重置">
    </form>
  </body>
</html>
```

Ex9_8.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<%@ page import="java.sql.*" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+
```

```
path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>添加记录</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->
  </head>
  <body>
   <%
        Connection conn;
        Statement stmt;
        ResultSet rs;
        String no=request.getParameter("no");
        String name=request.getParameter("name");
        String author=request.getParameter("author");
        String pub=request.getParameter("pub");
        String time=request.getParameter("date");
        String price=request.getParameter("money");
        String cat=request.getParameter("cat");
        name=new String(name.getBytes("ISO8859-1"),"GB2312");
        author=new String(author.getBytes("ISO8859-1"),"GB2312");
        pub=new String(pub.getBytes("ISO8859-1"),"GB2312");
        cat=new String(cat.getBytes("ISO8859-1"),"GB2312");
        float m=Float.parseFloat(price);
        try{
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
        conn = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;DatabaseName=BOOK",
        "sa","sa");
        stmt=conn.createStatement();
        String sql="insert into bookinfo values('';
        sql+=no+"','"+name+"','"+author+"','"+pub+"','"+time+"',"+price+",'"+price+"')";
        stmt.executeUpdate(sql);
        System.out.println("success");
        rs=stmt.executeQuery("select * from bookinfo");
        out.print("<table border>");
```

```
                out.print("<tr>");
                   out.print("<th>"+"编号");
                   out.print("<th>"+"书名");
                   out.print("<th>"+"作者");
                   out.print("<th >"+"出版社");
                   out.print("<th >"+"出版日期");
                   out.print("<th >"+"单价");
                   out.print("<th >"+"图书分类");
                out.print("</tr>");
                while(rs.next())
                {
                out.print("<tr>");
                   out.print("<td >"+rs.getString(1)+"</td>");
                   out.print("<td >"+rs.getString("name")+"</td>");
                   out.print("<td>"+rs.getString("author")+"</td>");
                   out.print("<td>"+rs.getString(4)+"</td>");
                   out.print("<td >"+rs.getString("time")+"</td>");
                   out.print("<td>"+rs.getFloat(6)+"</td>");
                   out.print("<td>"+rs.getString(7)+"</td>");
                out.print("</tr>");
                }
              }
          catch(ClassNotFoundException e1){}
          catch(SQLException e2){out.println(e2);}
            %>
        </body>
    </html>
```

9.5　删除记录

采用 delete 语句可以在表中删除记录。其结构非常简单。删除记录的 SQL 语句格式如下。

```
String sql= "delete  表名  where    条件  ";
Statement stmt=null;
stmt.executeUpdate( sql );
```

where 子句为可选项。用它来限制 delete 语句删除的行数。如果不写 where 子句，则表中的所有行都会被删除。通过使用 where 子句，可以指定要想删除记录所必须满足的条件，只有满足条件的记录才能被删除。

例 9_9：删除 bookinfo 数据库表中价格在 30 元以下的记录。

```
<%@ page language="java" import="java.sql.*" pageEncoding="GB2312"%>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+
path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>My JSP 'Ex9_9.jsp' starting page</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
    -->
  </head>
  <body>
   <%
        Connection conn;
        Statement stmt;
        ResultSet rs;
        try{Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();
         conn = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;DatabaseName=
         BOOK","sa","sa");
         stmt=conn.createStatement();
         String sql="delete bookinfo where price<=30";
         stmt.executeUpdate(sql);
         System.out.println("success");
         rs=stmt.executeQuery("select * from bookinfo");
         out.print("<table border>");
         out.print("<tr>");
            out.print("<th>"+"编号");
            out.print("<th>"+"书名");
            out.print("<th>"+"作者");
            out.print("<th >"+"出版社");
            out.print("<th >"+"出版日期");
            out.print("<th >"+"单价");
            out.print("<th >"+"图书分类");
         out.print("</tr>");
         while(rs.next())
```

```
                {
                out.print("<tr>");
                   out.print("<td >"+rs.getString(1)+"</td>");
                   out.print("<td >"+rs.getString("name")+"</td>");
                   out.print("<td>"+rs.getString("author")+"</td>");
                   out.print("<td>"+rs.getString(4)+"</td>");
                   out.print("<td >"+rs.getString("time")+"</td>");
                   out.print("<td>"+rs.getFloat(6)+"</td>");
                   out.print("<td>"+rs.getString(7)+"</td>");
                out.print("</tr>");
                }
              }
          catch(ClassNotFoundException e1){}
          catch(SQLException e2){out.println(e2);}
           %>
        </body>
    </html>
```

9.6　数据库连接池

在实际应用开发中，特别是在 Web 应用系统中，如果 JSP、Servlet 或 EJB 使用 JDBC 直接访问数据库中的数据，每一次数据访问请求都必须经历建立数据库连接、打开数据库、存取数据和关闭数据库连接等步骤。而连接并打开数据库是一件既消耗资源又费时的工作，如果频繁地进行这种数据库操作，系统的性能必然会急剧下降，甚至会导致系统崩溃。数据库连接池技术是解决这个问题最常用的方法，在许多应用程序服务器（如 Weblogic，WebSphere，JBoss）中，基本上都提供了这项技术，不用自己编程，因此，深入了解这项技术是非常必要的。

数据库连接池负责分配、管理和释放数据库连接，它允许程序重复使用一个现有的数据库连接，而不用重新建立数据库连接，这样可以大量减少所需的新连接，并且可以避免因没有释放数据库连接而引起的数据库连接错误。数据库连接池的优势在于它可以明显提高数据库操作的性能。

数据库连接池技术的思想非常简单，将数据库连接作为对象存储在一个 Vector 对象中，一旦数据库连接被建立后，不同的数据库访问请求就可以共享这些连接。这样，通过重复使用这些已经建立的数据库连接，可以解决上面提到的问题，极大地节省了系统资源和时间。

数据库连接池的主要操作如下。

（1）建立数据库连接池对象（服务器启动）。

（2）按照事先指定的参数创建初始数量的数据库连接，即空闲连接的数量。

（3）对于一个数据库访问请求，直接从数据库连接池中得到一个连接。如果数据库连接池对象中没有空闲连接，且连接数没有达到最大（即最大活跃连接数），创建一个新的数据库连接。

（4）存取数据。

（5）关闭数据库，释放所有数据库连接（此时关闭数据库，并非真正关闭，而是将数据库连接放入空闲队列中，如实际空闲连接数大于初始空闲连接数则释放连接）。

（6）释放数据库连接池对象（服务器停止、维护期间，释放数据库连接池对象，并释放所有连接）。

本书以 Tomcat 服务器自带的数据库连接池为例介绍配置和使用数据库连接池的方法。其步骤如下。

第 1 步：在%TOMCAT_HOME%\conf\server.xml 文件中<Host>…</Host>内添加配置代码（%TOMCAT_HOME%是指 Tomcat 的安装路径）如下所示。

```
<Context path="/Database_Prj" docBase="Database_Prj" debug="0"
        reloadable="true" crossContext="true" >

<Logger className="org.apache.catalina.logger.FileLogger"
        directory="logs"    prefix="localhost_quality_log." suffix=".txt"
        timestamp="true"/>
    <Resource      name="jdbc/connectDB"
                   auth="Container"
                   type="javax.sql.DataSource">
    </Resource>
    <ResourceParams name="jdbc/connectDB">
        <parameter>
            <name>maxActive</name>
            <value>100</value>
        </parameter>
        <parameter>
            <name>maxIdle</name>
            <value>30</value>
        </parameter>
        <parameter>
            <name>maxWait</name>
            <value>10000</value>
        </parameter>
        <parameter>
            <name>removeAbandoned</name>
            <value>true</value>
        </parameter>
        <parameter>
            <name>removeAbandonedTimeout</name>
            <value>60</value>
        </parameter>
        <parameter>
            <name>logAbandoned</name>
```

```
            <value>false</value>
        </parameter>
        <parameter>
            <name>factory</name>
            <value>org.apache.commons.dbcp.BasicDataSourceFactory
            </value>
        </parameter>
        <parameter>
            <name>driverClassName</name>
            <value>com.microsoft.jdbc.sqlserver.SQLServerDriver</value>
        </parameter>
        <parameter>
            <name>URL</name>
            <value>jdbc:sqlserver://localhost:1433;DatabaseName=BOOK
            </value>
        </parameter>
        <parameter>
            <name>username</name>
            <value>sa</value>
        </parameter>
        <parameter>
            <name>password</name>
            <value>sa</value>
        </parameter>
    </ResourceParams>
</Context>
```

对这段代码中各个部分的说明如下。

- <Context>标签的各种属性如下。
  path 属性指定项目所在的路径，以“/”开始。
  docBase 属性指定文件的根目录。
  reloadable 属性指定当网页被更新时是否重新编译，值为 true 或 false。
- <Resource>标签的 name 属性定义资源的名字，在本例中起名为 jdbc/connectDB。
- <ResourceParams>标签的 name 属性定义资源的名字。
- <parameter>标签被用来定义各种参数。<name>定义参数的名字；<value>定义参数的值。具体如下。
  maxActive 指定连接池的最大数据库连接数。若为 0 表示无限制。
  maxIdle 指定数据库连接的最长空闲时间。超过此空闲时间，数据库连接将被标记为不可用，然后被释放。若为 0 表示无限制。
  maxWait 指定最长建立连接等待时间。超过此时间将接到异常。若为−1 表示无限制。
  removeAbandoned 指定是否回收被遗弃的数据库连接到连接池中。被遗弃的数据库连接一般是忘记释放的数据库连接。值为 true 或 false。

removeAbandonedTimeout 指定数据库连接经过多长时间不用则被视为被遗弃而被回收到连接池中。时间单位为秒。

logAbandoned 指定是否将被遗弃的数据库连接的回收记入日志。值为 true 或 false。

driverClassName 指定连接数据库的驱动程序。在本例中是连接 SQL Server 数据库，其值为 com.microsoft.jdbc.sqlserver.SQLServerDriver。

URL 指定连接数据库的 URL 字符串。在本例中是连接 SQL Server 数据库的 BOOK 数据库，其值为 jdbc:sqlserver://localhost:1433;DatabaseName=BOOK。

username 指定连接数据库的用户名。

password 指定连接数据库的密码。

第 2 步：修改所建立项目所在文件夹下的 web.xml 文件，即在%TOMCAT_HOME%\webapps\Database_Prj\WEB_INF\web.xml 文件中添加如下代码。

```
<resource-ref>
        <description>connectDB test</description>
        <res-ref-name>jdbc/connectDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
</resource-ref>
```

本步的目的实际上是在项目中指定数据库连接池资源的引用方式。<res-ref-name>标签指出引用资源的名字，在本例中需要引用的就是在第 1 步中定义的数据库连接池 jdbc/connectDB。<res-type>标签指出该资源的类型，在本例中为“javax.sql.DataSource”。

第 3 步：编写测试代码，如下所示。

```
<%@ page language="java" import="java.sql.*" pageEncoding="GB2312"%>
<%@ page import="javax.naming.*,javax.sql.DataSource" %>
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+
path+"/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">
    <title>数据库连接池的使用</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
    <meta http-equiv="description" content="This is my page">
    <!--
    <link rel="stylesheet" type="text/css" href="styles.css">
```

```
    -->
</head>
<body>
  <%
        Context initCtx=null;
        Connection conn=null;
        Statement stmt=null;
        ResultSet rs=null;
        try{
          initCtx=new InitialContext();
          if(initCtx==null) throw new Exception("没有匹配的环境!");
          DataSource ds=(DataSource)initCtx.lookup("java:comp/env/jdbc/connectDB");
          if(ds==null)throw new Exception("没有匹配的数据库！ ");
          conn=ds.getConnection();
          out.println("success");
          stmt=conn.createStatement();
          rs = stmt.executeQuery("SELECT * FROM bookinfo");
          out.print("<table border>");
          out.print("<tr>");
            out.print("<th >"+"编号");
            out.print("<th>"+"书名");
            out.print("<th >"+"作者");
            out.print("<th >"+"出版社");
            out.print("<th >"+"出版日期");
            out.print("<th >"+"单价");
            out.print("<th>"+"图书分类");
          out.print("</tr>");
          while(rs.next())
          {
           out.print("<tr>");
           out.print("<td >"+rs.getString(1)+"</td>");
           out.print("<td >"+rs.getString("name")+"</td>");
           out.print("<td>"+rs.getString("author")+"</td>");
           out.print("<td>"+rs.getString(4)+"</td>");
           out.print("<td >"+rs.getString("time")+"</td>");
           out.print("<td>"+rs.getFloat(6)+"</td>");
           out.print("<td>"+rs.getString(7)+"</td>");
         out.print("</tr>");
          }
       }
      catch(SQLException e2){out.println(e2);}
       catch(Exception ee){
          System.out.println("connect db error:"+ee.getMessage());
       }
```

```
            finally{
              if(rs!=null) rs.close();
              if(stmt!=null) stmt.close();
              if(conn!=null) conn.close();
              if(initCtx!=null)initCtx.close();
             }
        %>
    </body>
</html>
```

9.7　数据库在项目中的应用

前面几节讲了数据库操作的基本概念，如数据源、JDBC、JDBC-ODBC 桥接器；然后又讲了最常见的数据库操作，如查询、修改、添加、删除数据记录等。为了解决数据并发执行问题，又讲了数据库连接池的原理与应用。现在以实际人力资源项目为例，结合 MVC 开发模式，讲解数据库连接池在项目中的应用。

为了使数据库操作更加方便，便于系统开发与维护，提高数据的执行效率，首先建立数据库操作接口。代码如下所示。

```
package com.etop.bank.comm.dbcom;
import java.sql.Connection;
import java.sql.ResultSet;
import java.util.List;

/**
 * @author db2admin
 *  需要通过 DataBaseFactory  的 getDBObject()静态方法来获取其实例化操作
*/
public interface DataBaseBsic {

    /**
     *  更新数据库，如修改、添加、删除;成功则返回 true,失败则返回 false
     * **/
    public boolean executeUpdate(String sql);

    /**
     *  查询数据库，返回结果集的 List 对象
     * **/
    public ResultSet executeQuery(String sql);
        public Connection getDBConnection();
    /*
     设置数据库，用于调试或初始化
```

```
        public void setDataBase(String DataBaseName);
        */
}
```

在前面建立了数据库操作接口，现在建立数据库操作工厂类来实现该接口，并且封装数据库操作所需要的所有方法，如修改方法、添加方法、删除方法等。代码如下所示。

```
package com.etop.bank.comm.dbcom;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;
import java.util.ListIterator;
import javax.naming.Context;
import javax.naming.NamingException;
import javax.sql.DataSource;

public final class DataBaseBsicFactory{
        private DBOption dbop=new DBOption();
        private DataBaseBsicFactory(){}

        /**
        * 获取工厂类
        * **/
        public static DataBaseBsicFactory getFactory(){
        return new DataBaseBsicFactory();
        }

        /**
         * 获取数据库操作对象，对 DataBaseBsic 进行实例化
         * **/
        public DBOption getDBObject(){
            return dbop;
        }
        /*业务实体*/
            private final class DBOption implements DataBaseBsic{
            private String JNDI="jdbc/";
            private ResultSet rs=null;
            private Statement sta=null;
            private Context ctx = null;
            private DataSource datasource = null;
            private Connection connection = null;
```

```
/*
 * 设置数据库，用于调试或初始化
 **/
private void setDataBase(String DataBaseName) {
     JNDI=JNDI+DataBaseName;
}
/*从数据源获得一个连接
 * */
private Connection getConnection(){
    try {
            ctx = new javax.naming.InitialContext();
        } catch (NamingException e) {
            // TODO 自动生成 catch 块
            e.printStackTrace();
        }
        try {
datasource=(javax.sql.DataSource)ctx.lookup("java:comp/env/"+JNDI);// 从 JNDI 获取 ds
        } catch (NamingException e1) {
            e1.printStackTrace();
        }
        try {
    connection = datasource.getConnection();// 从数据源获得一个连接
        } catch (SQLException e2) {
             e2.printStackTrace();
             return null;
        }
        return connection;

    }
    /*
     * 用于系统数据库操作
     * */
    private DBOption(){
          setdataBase("DB");
          getConnection();
    }

    /*
  * 执行数据库的修改、添加、删除操作;成功则返回 true,失败则返回 false
     * */
    private boolean execute(String sql){
        int i=-1;
        boolean flag=false;
        try {
```

```
            sta=connection.createStatement();
            i=sta.executeUpdate(sql);
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }

        if(i>0)flag=true;
        return flag;
    }

    /*
     *执行 select 性操作
     * */
    private ResultSet select(String sql){
        try {
            sta=connection.createStatement();
            rs=sta.executeQuery(sql);
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
            return rs;
      }

     /**
      * 更新数据库，如修改、添加、删除;成功则返回 true,失败则返回 false
      * **/
     public boolean executeUpdate(String sql) {
           boolean flag=false;
           flag=execute(sql);
           return flag;
     }
     /**
      * 查询数据库，返回结果集的 List 对象
      * **/
     public ResultSet executeQuery(String sql) {
           ResultSet rs=null;
           rs=select(sql);
           if(rs==null){
              System.out.println("null pointer");
              return null;
           }
           return rs;
```

```
        }
        /*
         * 用于满足其他操作 如 PreoaredStatement 等
         */
        public Connection getDBConnection() {
            Connection con= getConnection();
            if(con==null)return null;
            return con;
        }
    }
}
```

此数据库操作在简单数据库操作的基础之上进行了封装。先定义数据库操作接口 DataBaseBsic，在接口中定义操作方法，例如，对数据库的查询定义 public ResultSet execute Query（String sql）方法，对数据库的更新操作，如修改、添加、删除操作，则定义 public boolean executeUpdate（String sql）方法。然后又写接口实现类，该类是一个工厂类 Data BaseBsicFactory。若要调用实现类里的查询或更新数据库方法，则需要先通过 getFactory()方法获得该工厂类，通过工厂类的 getDBObject()方法获得接口实现的实体，再通过实现实体调用查询或更新数据库方法。如在第 8 章讲解用户管理模块实现时，在用户管理数据访问对象 DAO 实现中，对数据库的操作是通过定义数据库操作对象 db 实现的，例如：

```
private DataBaseBsic db=DataBaseBsicFatory.getFactory().getDBObject();
```

然后通过 db 调用获得数据库连接方法 getDBConnection()方法，对数据库的修改、添加、删除则通过 db 调用 executeUpdate(String sql)方法。这种采用工厂类的方法操作数据库的好处是使得数据库操作不是通过创建对象实现的，而是通过接口调用，避免数据库访问的强耦合性。

在工厂类 DataBaseBsicFactory 中有一个方法 getDBConnection()，主要是获取数据库连接，获取连接是采用 JNDI 数据库连接池实现的。

为了能够正常执行数据库连接池语句，需要在 Tomcat6.0conf 目录下的 context.xml 文件中添加如下代码以确定该数据库连接池的名字、数据类型、驱动名字、URL、用户名、密码等信息。

```
<Resource
auth="Container"
name="jdbc/DB"
type="javax.sql.DataSource"
driverClassName="com.mysql.jdbc.Driver"
URL="jdbc:mysql://localhost/bank"
maxIdle="60"
maxActive="1000"
maxWait="5000"
username="root"
password="123"
/>
```

为了在项目启动时便加载数据库连接池，需要在 web.xml 文件中添加如下代码以找到连接池源，从而可以加载和初始化连接池内容。

```
<resource-ref>
<res-ref-name>jdbc/DB</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

9.8　实训操作

在此次实训中，主要是要熟练掌握 JSP,Servlet,JavaBean 如何采用 JDBC 连接数据库，并且要熟练掌握在 Tomcat 中配置连接池的方法。请采用数据库连接池的方法完成人力资源管理系统中制度管理的制度维护功能。该功能包括添加制度名称、适用范围、制度状态、制度编号、开始时间等信息，然后再按制度名称、制度编号等查询条件查出制度列表，还包括编辑删除该制度。

习　　题

1．创建一个 Student 用户表，包含学号、姓名、专业、住址、特长等相关信息。创建 JSP 页面连接数据库（选用 JSP 调用 JavaBean 连接数据库更好），通过 SQL 语句添加一条新的记录（040501、张大伟、表演系、托普学院 1—406 寝、唱歌）；并访问数据库显示所有记录。

2．创建一个 Student 用户表，包含学号、姓名、专业、住址、特长等相关信息。采用数据库连接池技术连接数据库，完成根据姓名查询记录操作。

3．创建一个 Student 用户表，包含学号、姓名、专业、住址、特长等相关信息。采用 JDBC 技术连接数据库，完成根据某一个学号删除记录操作。

4．创建一个 Student 用户表，包含学号、姓名、专业、住址、特长等相关信息。请根据学号修改该学生的相关信息。

反侵权盗版声明